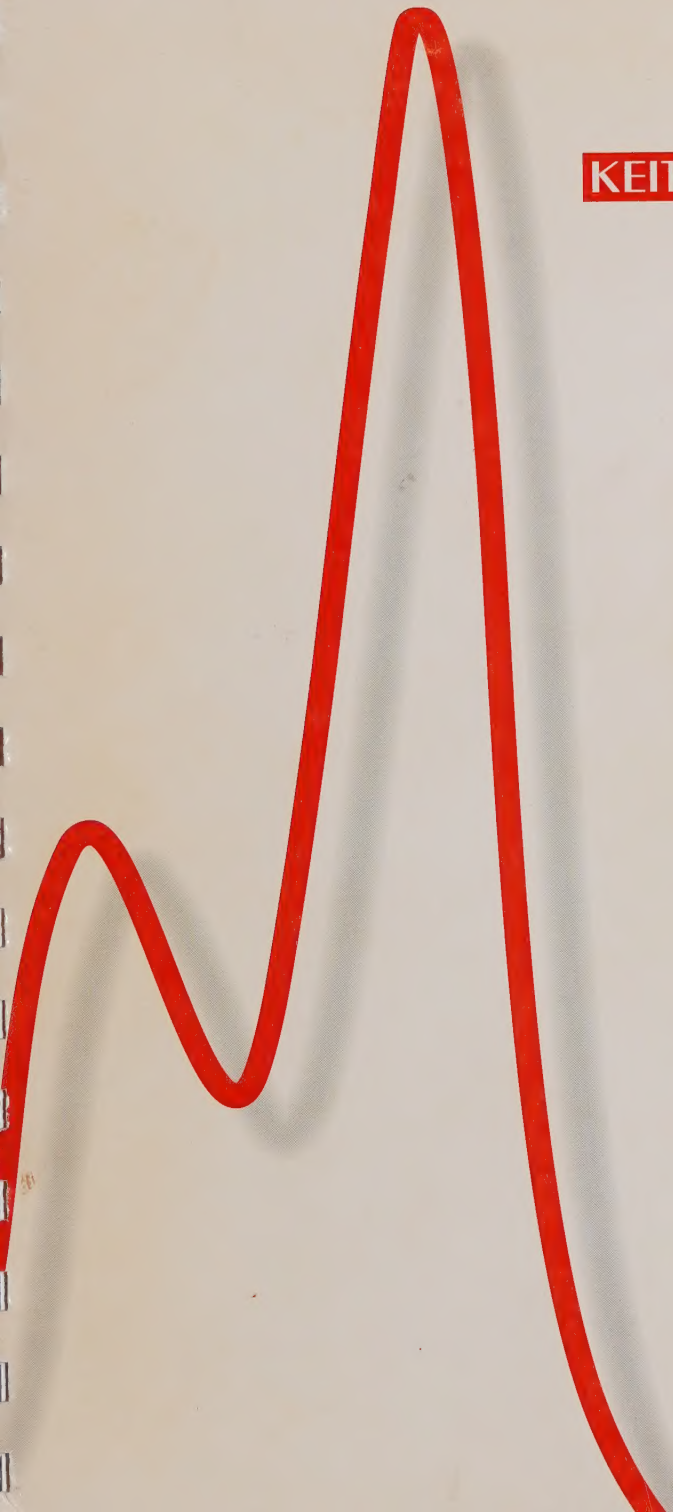


KEITHLEY METRABYTE

DAS-1700 Series Function Call Driver

USER'S GUIDE

Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_07714

DAS-1700 Series
Function Call Driver
User's Guide

Revision A - September 1996
Part Number: 97540

The information contained in this manual is believed to be accurate and reliable. However, Keithley Instruments, Inc., assumes no responsibility for its use or for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent rights of Keithley Instruments, Inc.

KEITHLEY INSTRUMENTS, INC., SHALL NOT BE LIABLE FOR ANY SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RELATED TO THE USE OF THIS PRODUCT. THIS PRODUCT IS NOT DESIGNED WITH COMPONENTS OF A LEVEL OF RELIABILITY SUITABLE FOR USE IN LIFE SUPPORT OR CRITICAL APPLICATIONS.

Refer to your Keithley Instruments license agreement and Conditions of Sale document for specific warranty and liability information.

MetraByte is a trademark of Keithley Instruments, Inc. All brand and product names are trademarks or registered trademarks of their respective companies.

© Copyright Keithley Instruments, Inc., 1996.

All rights reserved. Reproduction or adaptation of any part of this documentation beyond that permitted by Section 117 of the 1976 United States Copyright Act without permission of the Copyright owner is unlawful.

Keithley MetraByte Division

Keithley Instruments, Inc.

440 Myles Standish Blvd. Taunton, MA 02780

Telephone: (508) 880-3000 • FAX: (508) 880-0179

Table of Contents

Preface

1 Getting Started

Overview 1-1
Summary of Functions 1-2
Programming Flow Diagrams 1-7
Preliminary Steps for All Operations 1-7
Steps for an Analog Input Operation 1-8
Steps for an Analog Output Operation (DAS-1700ST-DA Series, DAS-1702HR-DA Boards) 1-15
Steps for an Analog Output Operation (DAS-1700AO Series Boards) 1-18
Steps for a Digital Input Operation 1-26
Steps for a Digital Output Operation 1-29
Getting Help 1-31

2 Available Operations

System Operations 2-1
Initializing the Driver 2-2
Initializing a Board 2-2
Retrieving Revision Levels 2-3
Handling Errors 2-3
Analog Input Operations 2-4
Operation Modes 2-4
Single Mode 2-4
Interrupt Mode 2-5
DMA Mode 2-5
Frames 2-6
Memory Allocation and Management 2-10
Dimensioning Local Arrays 2-10
Dynamically Allocating Memory Buffers 2-10
Assigning the Starting Addresses 2-12
Gains and Ranges 2-12
Channels 2-14
Specifying Channels when Using Expansion Accessories 2-14

Specifying a Single Channel 2-16
Specifying a Group of Consecutive Channels 2-16
Specifying Channels in a Channel-Gain Queue 2-17
Conversion Modes 2-18
Paced Mode 2-18
Burst Mode 2-18
Burst Mode with SSH 2-19
Clock Sources 2-19
Pacer Clock 2-19
Internal A/D Pacer Clock 2-20
External Pacer Clock 2-21
Burst Mode Conversion Clock 2-22
Buffering Modes 2-23
Single-Cycle Mode 2-23
Continuous Mode 2-24
Triggers 2-24
Trigger Sources 2-24
Internal Trigger 2-24
External Analog Trigger 2-25
External Digital Trigger 2-28
Trigger Acquisition 2-29
Post-Trigger Acquisition 2-29
Pre-Trigger Acquisition 2-30
About-Trigger Acquisition 2-30
Hardware Gates 2-31
Analog Output Operations (ST-DA, HR-DA) 2-32
Operation Modes 2-33
Single Mode 2-33
Interrupt Mode 2-33
Frames 2-34
Memory Allocation and Management 2-35
Channels 2-36
Pacer Clock 2-37
Buffering Modes 2-39
Single-Cycle Mode 2-39
Continuous Mode 2-39
Analog Output Operations (AO) 2-40
Operation Modes 2-40
Single Mode 2-40
Interrupt Mode 2-40
DMA Mode 2-41
Recycle Mode 2-42

Frames 2-42
Memory Allocation and Management 2-44
Dimensioning Local Arrays 2-45
Dynamically Allocating Memory Buffers 2-45
Assigning the Starting Addresses 2-47
Analog Output Ranges 2-47
Channels 2-48
Specifying a Single Channel 2-48
Specifying Both Channels Using the Same Output Range 2-48
Specifying Both Channels Using Different Output Ranges (Channel-Gain Queue) 2-49
Pacer Clock 2-50
Internal D/A Pacer Clock 2-50
External Pacer Clock 2-51
Clock Used for an Analog Input Operation 2-52
Buffering Modes 2-53
Single-Cycle Mode 2-53
Continuous Mode 2-53
Triggers 2-54
Internal Trigger 2-54
External Analog Trigger 2-54
External Digital Trigger 2-54
Hardware Gates 2-55
Digital I/O Operations 2-56
Operation Modes 2-56
Single Mode 2-56
Interrupt Mode 2-57
Frames 2-58
Memory Allocation and Management 2-59
Digital Input Channel 2-60
Digital Output Channel 2-61
Pacer Clock 2-61
Buffering Modes 2-63
Single-Cycle Mode 2-64
Continuous Mode 2-64

3 Programming with the Function Call Driver

Programming Overview 3-2
C/C++ Programming Information 3-3
Dynamically Allocating and Assigning Memory Buffers . . . 3-3
Allocating a Single Memory Buffer 3-4
Allocating Multiple Memory Buffers 3-4
Accessing the Data 3-5
Dimensioning and Assigning Local Arrays 3-6
Dimensioning a Single Array 3-6
Dimensioning Multiple Arrays 3-6
Creating a Channel-Gain Queue 3-7
Handling Errors 3-8
Programming in Microsoft C/C++ for Windows 3-9
Programming in Borland C/C++ for Windows 3-10
Pascal Programming Information 3-11
Dynamically Allocating and Assigning Memory Buffers . . 3-11
Allocating a Single Memory Buffer 3-11
Allocating Multiple Memory Buffers 3-13
Accessing the Data 3-14
Dimensioning and Assigning Local Arrays 3-14
Dimensioning a Single Array 3-14
Dimensioning Multiple Arrays 3-14
Creating a Channel-Gain Queue 3-15
Handling Errors 3-16
Programming in Borland Turbo Pascal for Windows 3-16
Visual Basic for Windows Programming Information 3-17
Dynamically Allocating and Assigning Memory Buffers . . 3-17
Allocating a Single Memory Buffer 3-17
Allocating Multiple Memory Buffers 3-18
Accessing Data from a Dynamically Allocated Memory Buffer with Fewer than 64 KB of Data 3-19
Accessing Data from a Dynamically Allocated Memory Buffer with More than 64 KB of Data 3-19
Accessing More than 64 KB of Data from a Dynamically Allocated Buffer 3-21
Dimensioning and Assigning Local Arrays 3-22
Dimensioning a Single Array 3-22
Dimensioning Multiple Arrays 3-22
Creating a Channel-Gain Queue 3-23
Converting 16-Bit Values (DAS-1700HR Series) 3-24
Handling Errors 3-25
Programming in Visual Basic for Windows 3-25

4 Function Reference

K_ADRead 4-7
K_BufListAdd 4-9
K_BufListReset 4-12
K_ClearFrame 4-14
K_CloseDriver 4-16
K_ClrAboutTrig 4-18
K_ClrADFreeRun 4-20
K_ClrContRun 4-22
K_DASDevInit 4-24
K_DAWriteGain 4-26
K_DIRead 4-29
K_DMAAlloc 4-31
K_DMAFree 4-34
K_DMAStart 4-36
K_DMAStatus 4-38
K_DMAStop 4-43
K_DOWrite 4-46
K_FormatChnGAry 4-48
K_FreeDevHandle 4-49
K_FreeFrame 4-51
K_GetADCommonMode 4-52
K_GetADConfig 4-54
K_GetADFrame 4-56
K_GetADMode 4-58
K_GetBurstTicks 4-60
K_GetClkRate 4-62
K_GetDAFrame 4-64
K_GetDevHandle 4-66
K_GetDIFrame 4-68
K_GetDOFrame 4-70
K_GetErrMsg 4-72
K_GetShellVer 4-73
K_GetVer 4-75
K_IntAlloc 4-77
K_IntFree 4-80
K_IntStart 4-81
K_IntStatus 4-83
K_IntStop 4-87
K_MoveArrayToBuf 4-89
K_MoveBufToArray 4-91
K_OpenDriver 4-93

K_RestoreChnGAry 4-95
K_SetAboutTrig 4-96
K_SetADCommonMode 4-98
K_SetADConfig 4-100
K_SetADFreeRun 4-102
K_SetADMode 4-104
K_SetADTrig 4-106
K_SetBuf 4-108
K_SetBufI 4-111
K_SetBurstTicks 4-113
K_SetChn 4-115
K_SetChnGAry 4-117
K_SetClk 4-120
K_SetClkRate 4-123
K_SetContRun 4-125
K_SetDITrig 4-127
K_SetDMABuf 4-129
K_SetExtClkEdge 4-131
K_SetG 4-133
K_SetGate 4-135
K_SetSSH 4-137
K_SetStartStopChn 4-139
K_SetStartStopG 4-141
K_SetSync 4-144
K_SetTrig 4-146
K_SetTrigHyst 4-148

A Error/Status Codes

B Data Formats

Converting Counts to Voltage B-1
Converting Voltage to Counts B-4
Specifying an Analog Output Value B-4
Specifying an Analog Trigger Level B-5
Specifying a Hysteresis Value B-6

Index

List of Figures

Figure 2-1. Frame-Based Operation 2-7
Figure 2-2. Example of Logical Channel Assignments 2-15
Figure 2-3. Analog Trigger Conditions 2-25
Figure 2-4. Using a Hysteresis Value 2-27
Figure 2-5. Digital Trigger Conditions 2-28
Figure 2-6. Digital Input Bits 2-60
Figure 2-7. Digital Output Bits 2-61

List of Tables

Table 1-1. Summary of Functions 1-2
Table 2-1. A/D Frame Elements 2-8
Table 2-2. Analog Input Ranges 2-13
Table 2-3. D/A Frame Elements (DAS-1700ST-DA Series and DAS-1702HR-DA) 2-34
Table 2-4. D/A Frame Elements (DAS-1700AO Series) . . . 2-43
Table 2-5. Analog Output Ranges (DAS-1700AO Series) . . 2-47
Table 2-6. DI Frame Elements 2-58
Table 2-7. DO Frame Elements 2-59
Table 3-1. Protected-Mode Memory Architecture 3-20
Table 4-1. Functions 4-2
Table 4-2. Data Type Prefixes 4-6
Table A-1. Error/Status Codes A-1
Table B-1. Span Values For Data Conversion Equations B-2

Preface

The *DAS-1700 Series Function Call Driver User's Guide* describes how to write programs for DAS-1700 Series boards using the DAS-1700 Series Function Call Driver.

The DAS-1700 Series Function Call Driver supports the following Windows®-based languages:

- Microsoft® C/C++ (Version 7.0 and higher)
- Borland® C/C++ (Version 4.5 and higher)
- Microsoft Visual Basic® for Windows (Version 3.0 and higher)
- Microsoft Visual C++™ (Version 1.52 and higher)
- Borland Turbo Pascal® for Windows (Version 1.0 and higher)

The manual is intended for application programmers using a DAS-1700 Series board in an IBM® PC AT® or compatible computer. It is assumed that you have read the *DAS-1700 Series User's Guide* to familiarize yourself with the features of the board, and that you have completed the appropriate hardware installation and configuration.

It is also assumed that you are experienced in programming in your selected language and that you are familiar with data acquisition principles.

The *DAS-1700 Series Function Call Driver User's Guide* is organized as follows:

- Chapter 1 contains a summary of the DAS-1700 Series Function Call Driver functions, flow diagrams illustrating the procedures used when programming each of the operations supported by the Function Call Driver, and instructions for getting help.

- Chapter 2 contains the background information needed to use the functions included in Chapter 4 of the DAS-1700 Series Function Call Driver.
- Chapter 3 contains a programming overview and language-specific information related to using the DAS-1700 Series Function Call Driver.
- Chapter 4 contains detailed descriptions of the DAS-1700 Series Function Call Driver functions, arranged in alphabetical order.
- Appendix A contains a list of the error codes returned by DAS-1700 Series Function Call Driver functions.
- Appendix B contains instructions for converting counts to voltage and for converting voltage to counts.

An index completes this manual.

Keep the following conventions in mind as you use this manual:

- References are made to the following series of boards in the DAS-1700 family
  - *DAS-1700 Series* - Includes the DAS-1701ST, DAS-1702ST, DAS-1701ST-DA, DAS-1702ST-DA, DAS-1702HR, DAS-1702HR-DA, DAS-1701AO, and DAS-1702AO boards.
  - *DAS-1700ST Series* - Includes the DAS-1701ST, DAS-1701ST-DA, DAS-1702ST, and DAS-1702ST-DA boards.
  - *DAS-1700HR Series* - Includes the DAS-1702-HR and DAS-1702HR-DA boards.
  - *DAS-1700ST-DA Series* - Includes the DAS-1701ST-DA and DAS-1702ST-DA boards.
  - *DAS-1700AO Series* - Includes the DAS-1701AO and DAS-1702AO boards.

  When a feature applies only to a particular board or to a particular series, the specific board name or series name is used.
- References to Windows apply to Windows 3.1, Windows 3.11 for Workgroups, and Windows 95. When a feature applies to a specific Windows version, the complete version name is used.

- References to Visual Basic apply to Microsoft Visual Basic for Windows. Similarly, references to C/C++ include the supported versions of Microsoft C/C++, Borland C/C++, and Microsoft Visual C++.
- Keyboard keys are represented in bold.

1

Getting Started

This chapter contains the following sections:

- **Overview** - a description of the DAS-1700 Series Function Call Driver.
- **Summary of Functions** - a brief description of the DAS-1700 Series Function Call Driver functions.
- **Programming Flow Diagrams** - illustrations of the procedures to follow when programming a DAS-1700 Series board using the DAS-1700 Series Function Call Driver.
- **Getting Help** - information on how to get help when installing or using the DAS-1700 Series Function Call Driver.

Overview

The DAS-1700 Series Function Call Driver is a library of data acquisition and control functions (referred to as the Function Call Driver or FCD functions). It is part of the Advanced Software Option (ASO) package that accompanies each DAS-1700 Series board. This package includes the following:

- Dynamic Link Libraries (DLLs) of FCD functions for Microsoft C/C++, Borland C/C++, Microsoft Visual C++, Microsoft Visual Basic for Windows, and Borland Turbo Pascal for Windows.
- Support files containing program elements, such as function prototypes and definitions of variable types, that are required by the FCD functions.

- Utility programs, running under DOS and Windows, that allow you to configure, calibrate, and test the functions of DAS-1700 Series boards.
- Language-specific example programs.
- ASO-Win95/32-bit disk, for creating 32-bit programs that run under Windows 95.

Before you use the Function Call Driver, make sure that you have installed the software, set up the board, and created a configuration file using the setup and installation procedures described in Chapter 2 of the *DAS-1700 Series User's Guide*.

Summary of Functions

Table 1-1 provides a brief description of the functions in the DAS-1700 Series Function Call Driver. For more detailed information about the functions, refer to Chapter 4.

Table 1-1. Summary of Functions

Type of Function	Name of Function	Description
Initialization	K_OpenDriver	Initializes a Function Call Driver.
	K_CloseDriver	Closes a Function Call Driver.
	K_GetDevHandle	Initializes a Keithley DAS board.
	K_FreeDevHandle	Frees a device handle.
	K_DASDevInit	Reinitializes a board.

Table 1-1. Summary of Functions (cont.)

Type of Function	Name of Function	Description
Operation	K_ADRead	Reads a single analog input value.
	K_DAWriteGain	Writes a single analog output value.
	K_DIRead	Reads a single digital input value.
	K_DOWrite	Writes a single digital output value.
	K_DMAStart	Starts a DMA-mode or recycle-mode operation.
	K_DMAStatus	Gets the status of a DMA-mode or recycle-mode operation.
	K_DMAStop	Stops a DMA-mode or recycle-mode operation.
	K_IntStart	Starts an interrupt-mode or recycle-mode operation.
	K_IntStatus	Gets the status of an interrupt-mode or recycle-mode operation.
	K_IntStop	Stops an interrupt-mode or recycle-mode operation.
Frame management	K_GetADFrame	Accesses a frame for an analog input operation.
	K_GetDAFrame	Accesses a frame for an analog output operation.
	K_GetDIFrame	Accesses a frame for a digital input operation.
	K_GetDOFrame	Accesses a frame for a digital output operation.
	K_FreeFrame	Frees a frame.
	K_ClearFrame	Sets all frame elements to their default values.

Table 1-1. Summary of Functions (cont.)

Type of Function	Name of Function	Description
Memory management	K_DMAAlloc	Dynamically allocates a memory buffer for a DMA-mode or recycle-mode operation.
	K_DMAFree	Frees a memory buffer that was dynamically allocated for a DMA-mode or recycle-mode operation.
	K_IntAlloc	Dynamically allocates a memory buffer for an interrupt-mode or recycle-mode operation.
	K_IntFree	Frees a memory buffer that was dynamically allocated for an interrupt-mode or recycle-mode operation.
	K_MoveArrayToBuf	Transfers data from a local integer array to a dynamically allocated memory buffer.
	K_MoveBufToArray	Transfers data from a dynamically allocated memory buffer to a local integer array.
Buffer address	K_SetBuf	Specifies the address of a local array (C/C++ or Pascal) or a dynamically allocated memory buffer (all supported languages) for an interrupt-mode or recycle-mode operation.
	K_SetBufI	Specifies the address of a local integer array (Visual Basic) for an interrupt-mode or recycle-mode operation.
	K_SetDMABuf	Specifies the address of a dynamically allocated memory buffer for a DMA-mode operation.
	K_BufListAdd	Adds a buffer or array to the list of multiple buffers or arrays.
	K_BufListReset	Clears the list of multiple buffers or arrays.
Buffering mode	K_SetContRun	Specifies continuous mode.
	K_ClrContRun	Specifies single-cycle mode.
Conversion mode	K_SetADFreeRun	Specifies burst mode.
	K_ClrADFreeRun	Specifies paced mode.
	K_SetSSH	Specifies burst mode with SSH (simultaneous sample-and-hold).

Table 1-1. Summary of Functions (cont.)

Type of Function	Name of Function	Description
Channel and gain	K_SetChn	Specifies a single channel.
	K_SetStartStopChn	Specifies the first and last channels in a group of consecutive channels.
	K_SetG	Specifies the gain for a group of consecutive channels.
	K_SetStartStopG	Specifies the first and last channels in a group of consecutive channels and the gain for all channels in the group.
	K_SetChnGAry	Specifies the starting address of a channel-gain queue.
	K_FormatChnGAry	Converts the format of a channel-gain queue.
	K_RestoreChnGAry	Restores a converted channel-gain queue.
	K_SetADCommonMode	Specifies the analog input common-mode ground reference.
	K_SetADConfig	Specifies the input channel configuration (differential or single-ended).
	K_SetADMode	Specifies the input range type (bipolar or unipolar).
	K_GetADCommonMode	Gets the analog input common-mode ground reference.
	K_GetADConfig	Gets the input channel configuration (differential or single-ended).
	K_GetADMode	Gets the input range type (bipolar or unipolar).

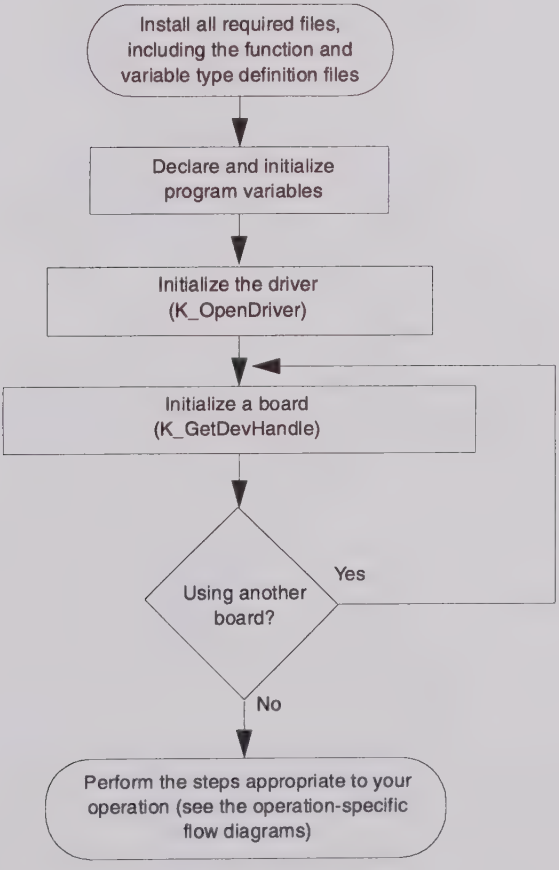
Table 1-1. Summary of Functions (cont.)

Type of Function	Name of Function	Description
Clock	K_SetClk	Specifies the pacer clock source.
	K_SetClkRate	Specifies the clock rate for the internal pacer clock.
	K_SetExtClkEdge	Specifies the active edge of an external pacer clock.
	K_SetBurstTicks	Specifies the burst mode conversion rate.
	K_GetClkRate	Gets the clock rate for the internal pacer clock.
	K_GetBurstTicks	Gets the burst mode conversion rate.
	K_SetSync	Specifies the operation to which an analog output operation is synchronized.
Trigger	K_SetTrig	Specifies the trigger source.
	K_SetADTrig	Sets up an external analog trigger.
	K_SetTrigHyst	Specifies the hysteresis value.
	K_SetDITrig	Sets up an external digital trigger.
	K_SetAboutTrig	Enables the about trigger and specifies the number of post-trigger samples.
	K_ClrAboutTrig	Disables the about trigger.
Gate	K_SetGate	Specifies the status of a hardware gate.
Miscellaneous	K_GetErrMsg	Gets the address of an error message string.
	K_GetVer	Gets revision numbers.
	K_GetShellVer	Gets the current DAS shell version.

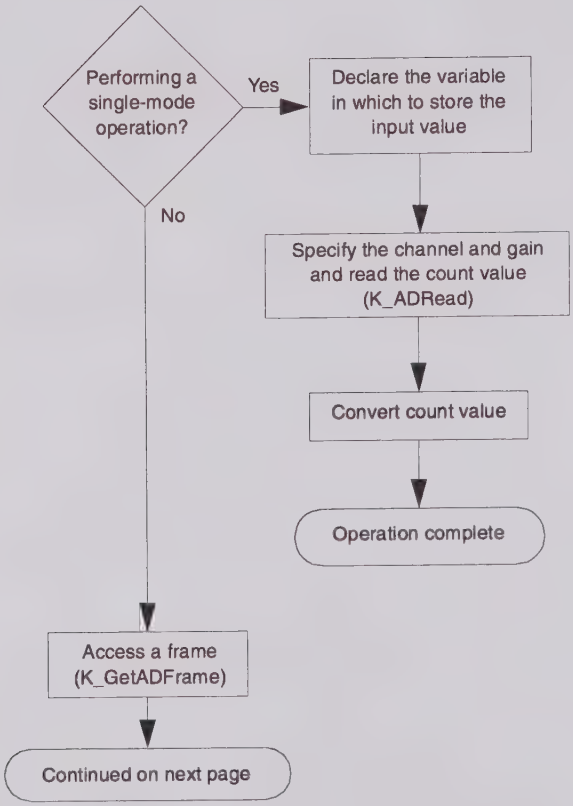
Programming Flow Diagrams

This section contains a series of programming flow diagrams illustrating the procedures to use when programming each of the operations supported by the DAS-1700 Series Function Call Driver. Although error checking is not shown in the flow diagrams, it is recommended that you check the error/status code returned by each function used in your program. Refer to Appendix A for a complete list of error codes.

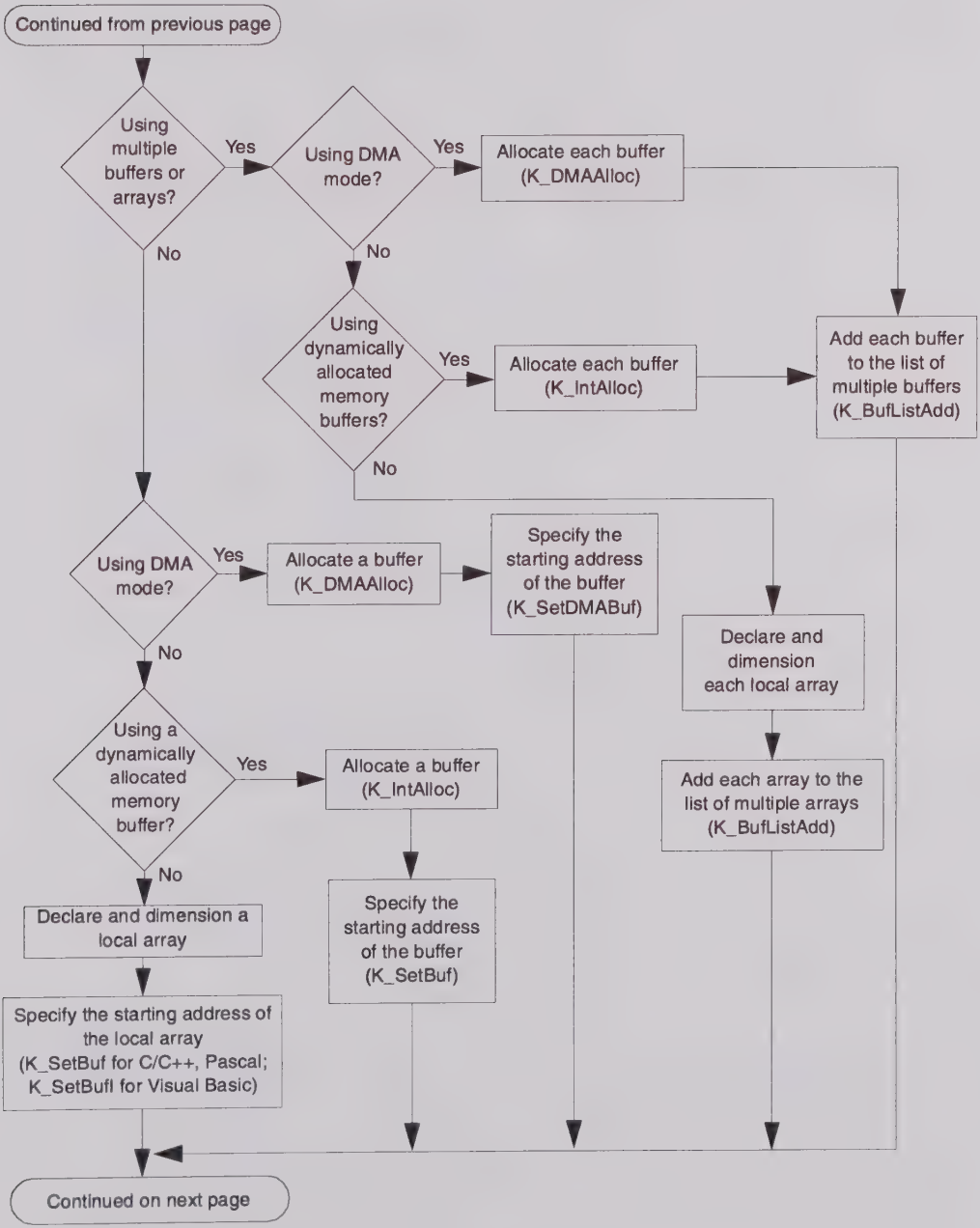
Preliminary Steps for All Operations


Install all required files, including the function and variable type definition files
Declare and initialize program variables
Initialize the driver (K_OpenDriver)
Initialize a board (K_GetDevHandle)
Using another board?
Yes
No
Perform the steps appropriate to your operation (see the operation-specific flow diagrams)

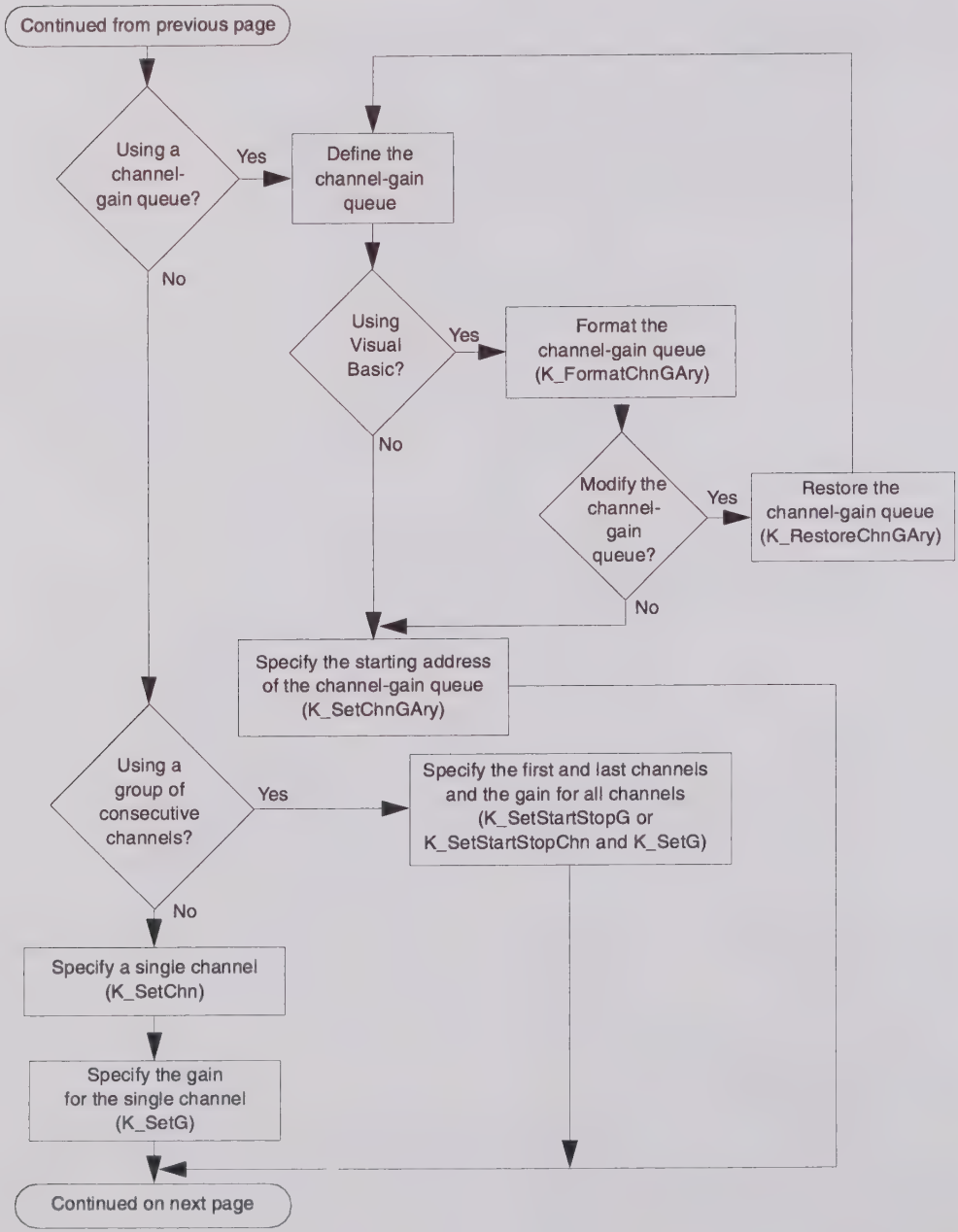
Steps for an Analog Input Operation


Performing a single-mode operation?
Yes
No
Declare the variable in which to store the input value
Specify the channel and gain and read the count value (K_ADRead)
Convert count value
Operation complete
Access a frame (K_GetADFrame)
Continued on next page

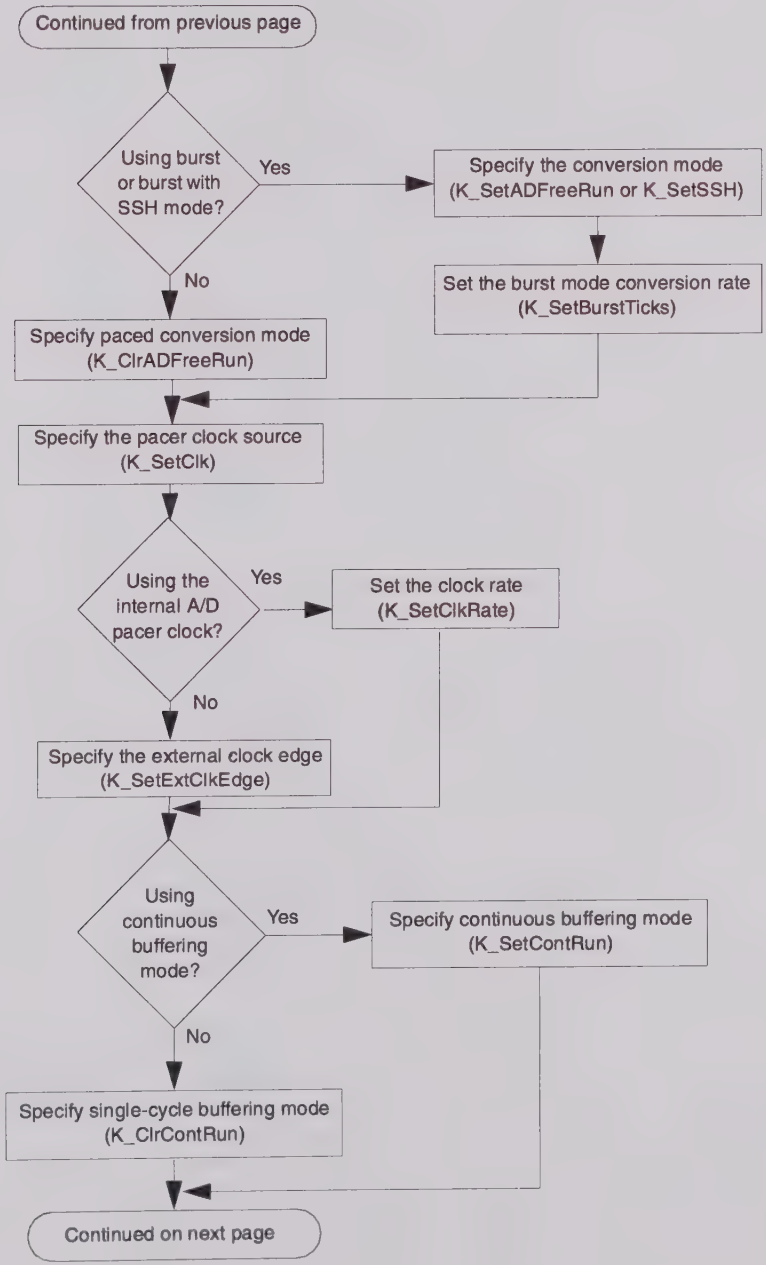
Steps for an Analog Input Operation (cont.)

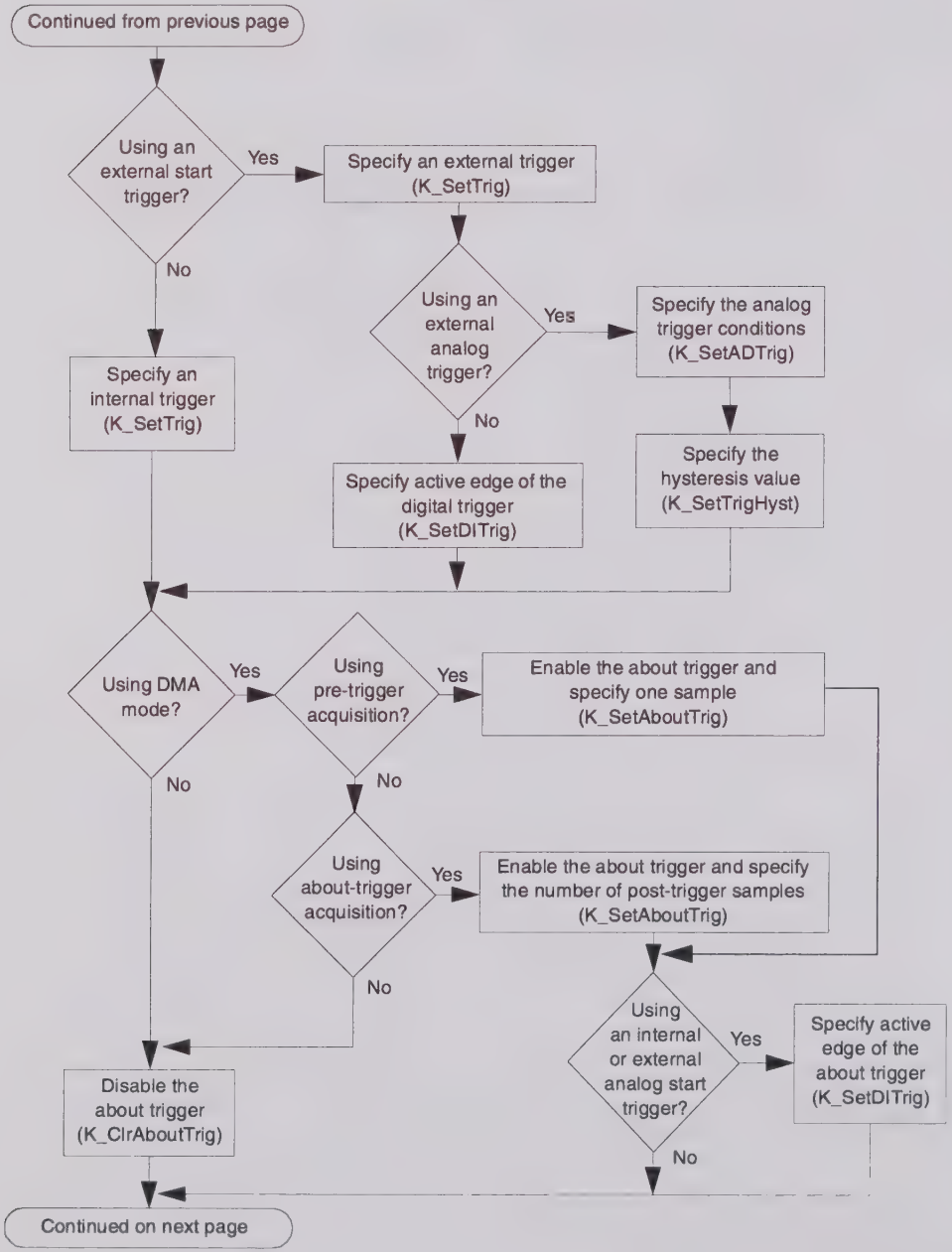

Continued from previous page
Using multiple buffers or arrays?
Yes
No
Using DMA mode?
Yes
No
Allocate each buffer (K_DMAAlloc)
Using dynamically allocated memory buffers?
Yes
No
Allocate each buffer (K_IntAlloc)
Add each buffer to the list of multiple buffers (K_BufListAdd)
Using DMA mode?
Yes
No
Allocate a buffer (K_DMAAlloc)
Specify the starting address of the buffer (K_SetDMABuf)
Declare and dimension each local array
Add each array to the list of multiple arrays (K_BufListAdd)
Using a dynamically allocated memory buffer?
Yes
No
Allocate a buffer (K_IntAlloc)
Specify the starting address of the buffer (K_SetBuf)
Declare and dimension a local array
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Continued on next page

Steps for an Analog Input Operation (cont.)

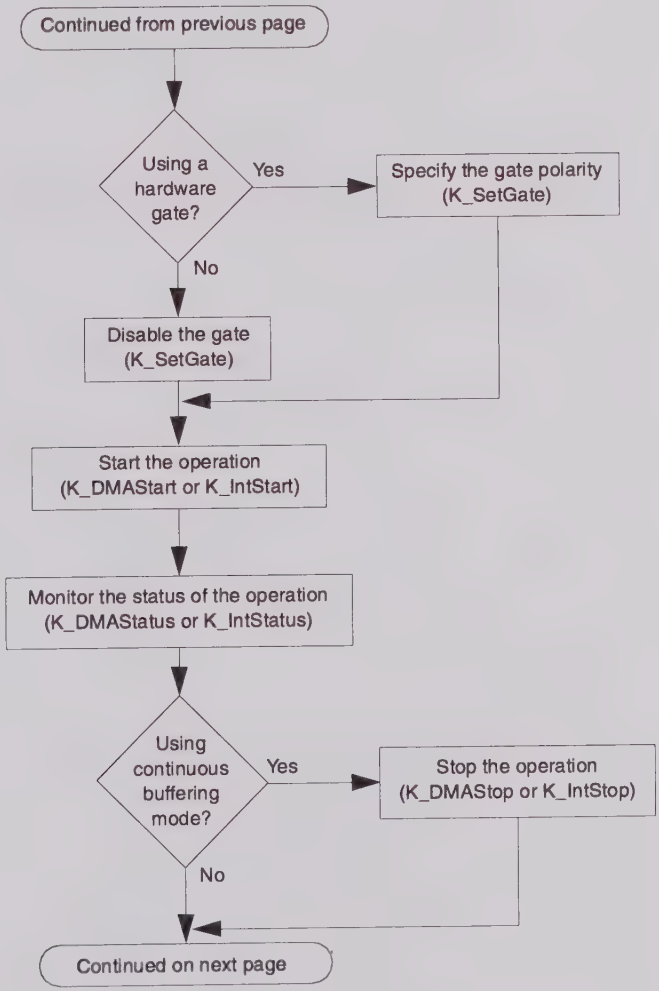

Continued from previous page
Using a channel-gain queue?
Yes
No
Define the channel-gain queue
Using Visual Basic?
Yes
No
Format the channel-gain queue (K_FormatChnGAry)
Modify the channel-gain queue?
Yes
No
Restore the channel-gain queue (K_RestoreChnGAry)
Specify the starting address of the channel-gain queue (K_SetChnGAry)
Using a group of consecutive channels?
Yes
No
Specify the first and last channels and the gain for all channels (K_SetStartStopG or K_SetStartStopChn and K_SetG)
Specify a single channel (K_SetChn)
Specify the gain for the single channel (K_SetG)
Continued on next page

Steps for an Analog Input Operation (cont.)

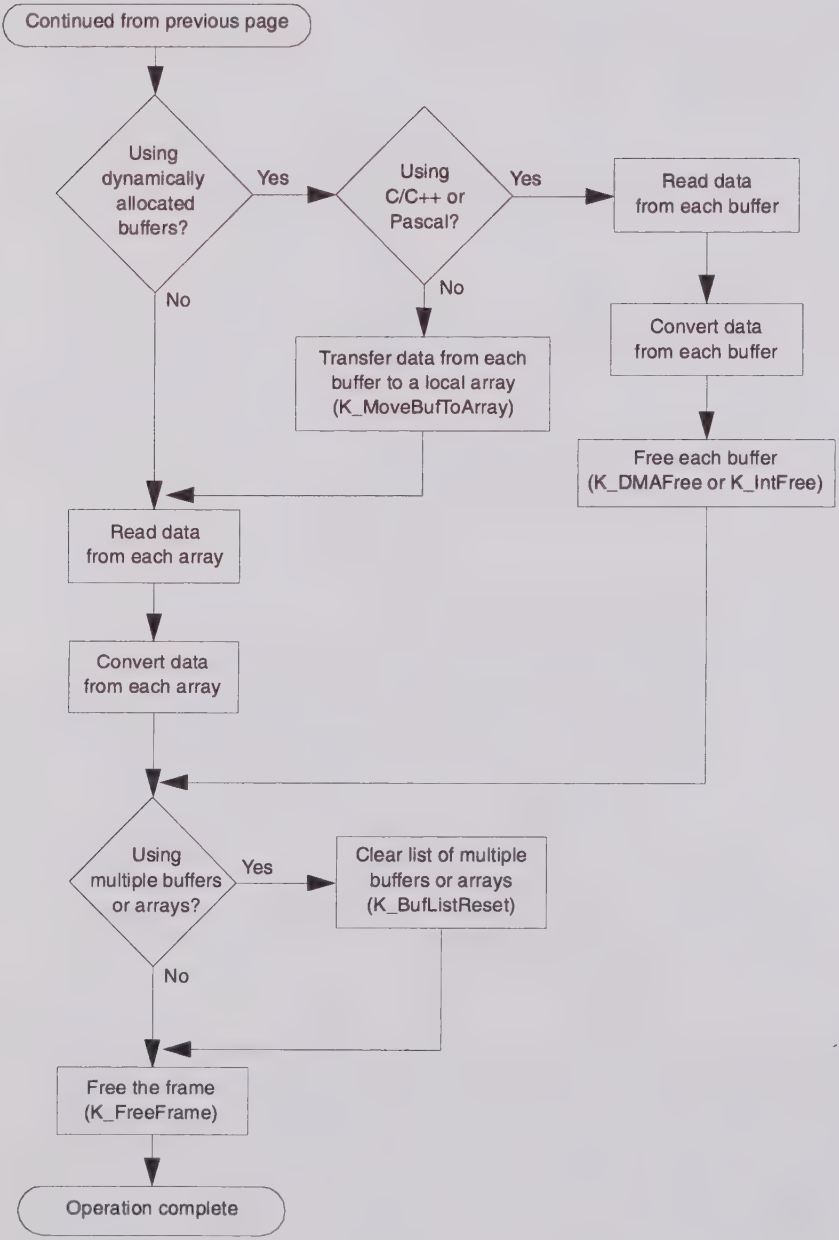

Continued from previous page
Using burst or burst with SSH mode?
Yes
No
Specify the conversion mode (K_SetADFreeRun or K_SetSSH)
Set the burst mode conversion rate (K_SetBurstTicks)
Specify paced conversion mode (K_ClrADFreeRun)
Specify the pacer clock source (K_SetClk)
Using the internal A/D pacer clock?
Yes
No
Set the clock rate (K_SetClkRate)
Specify the external clock edge (K_SetExtClkEdge)
Using continuous buffering mode?
Yes
No
Specify continuous buffering mode (K_SetContRun)
Specify single-cycle buffering mode (K_ClrContRun)
Continued on next page


Continued from previous page
Using an external start trigger?
Yes
No
Specify an external trigger (K_SetTrig)
Specify an internal trigger (K_SetTrig)
Using an external analog trigger?
Yes
No
Specify the analog trigger conditions (K_SetADTrig)
Specify the hysteresis value (K_SetTrigHyst)
Specify active edge of the digital trigger (K_SetDITrig)
Using DMA mode?
Yes
No
Using pre-trigger acquisition?
Yes
No
Enable the about trigger and specify one sample (K_SetAboutTrig)
Using about-trigger acquisition?
Yes
No
Enable the about trigger and specify the number of post-trigger samples (K_SetAboutTrig)
Using an internal or external analog start trigger?
Yes
No
Specify active edge of the about trigger (K_SetDITrig)
Disable the about trigger (K_ClrAboutTrig)
Continued on next page

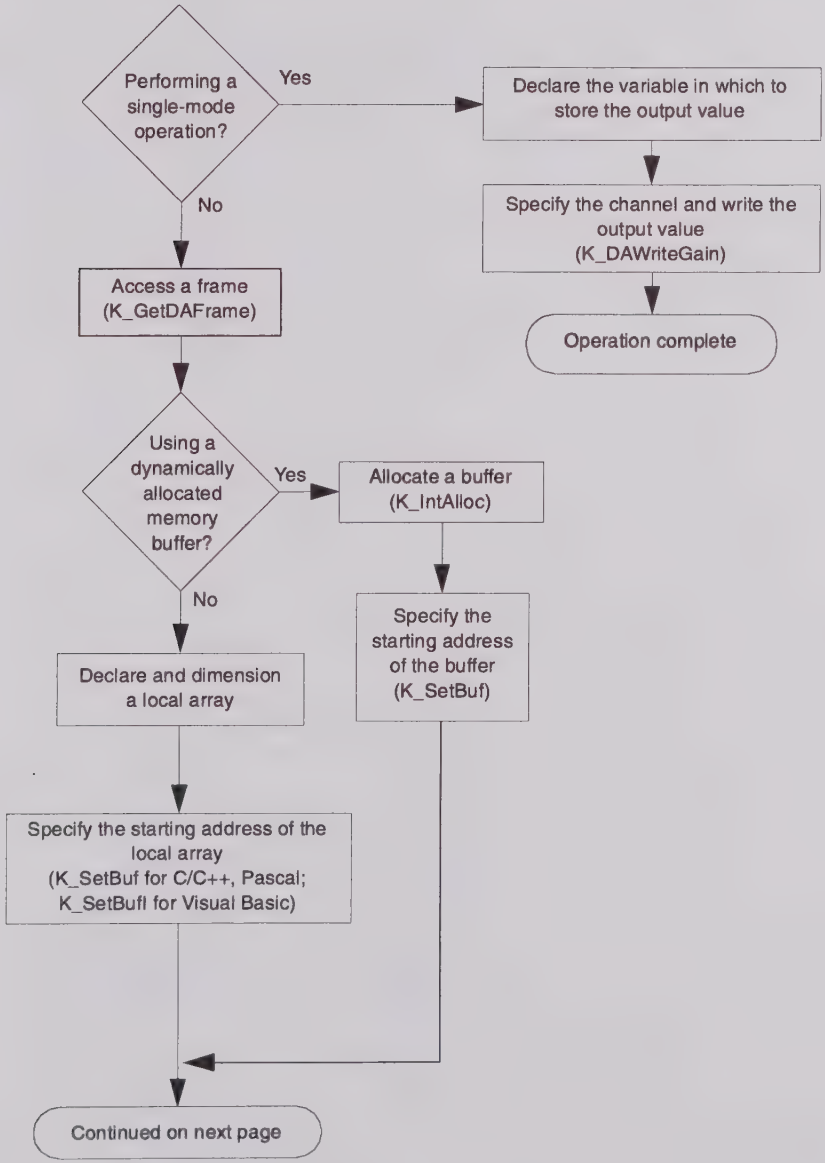
Steps for an Analog Input Operation (cont.)

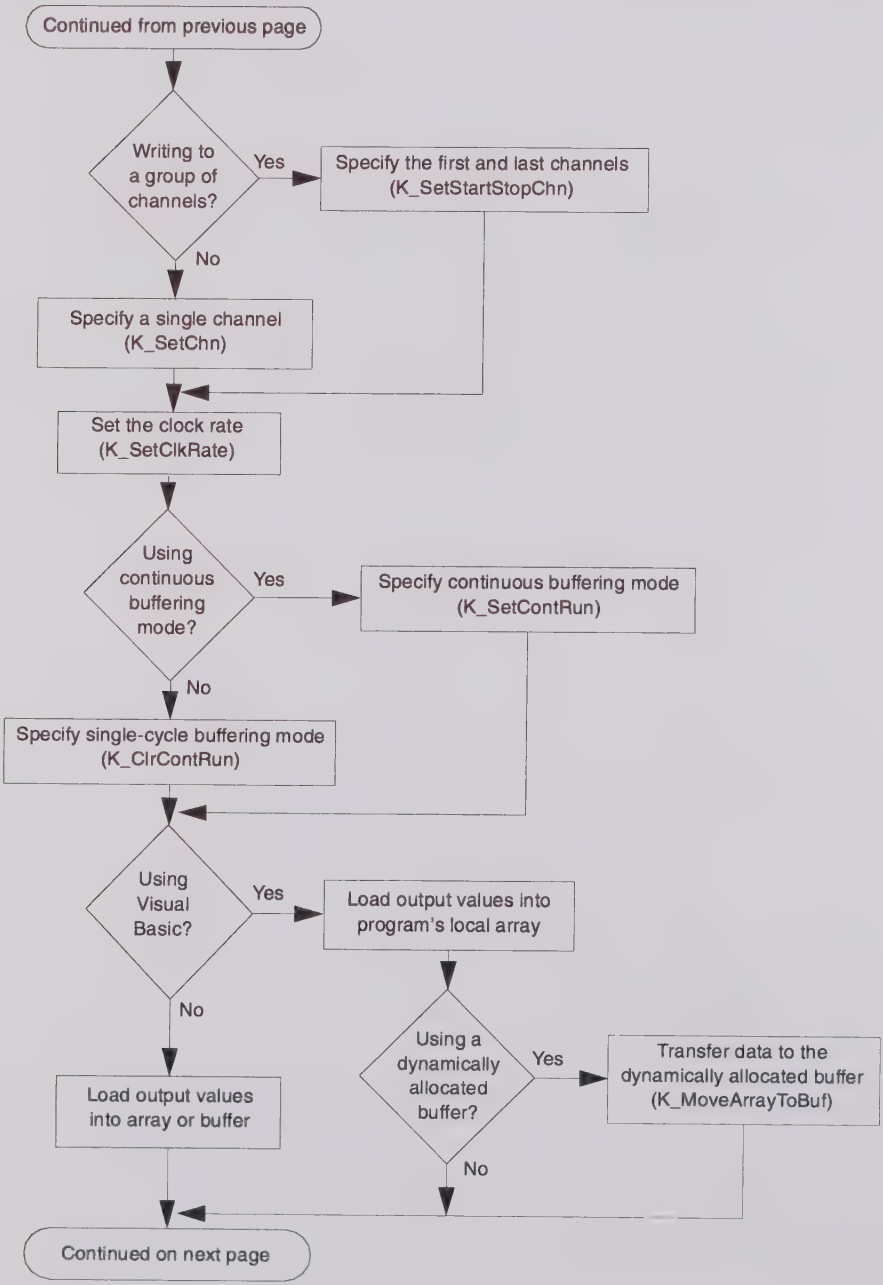


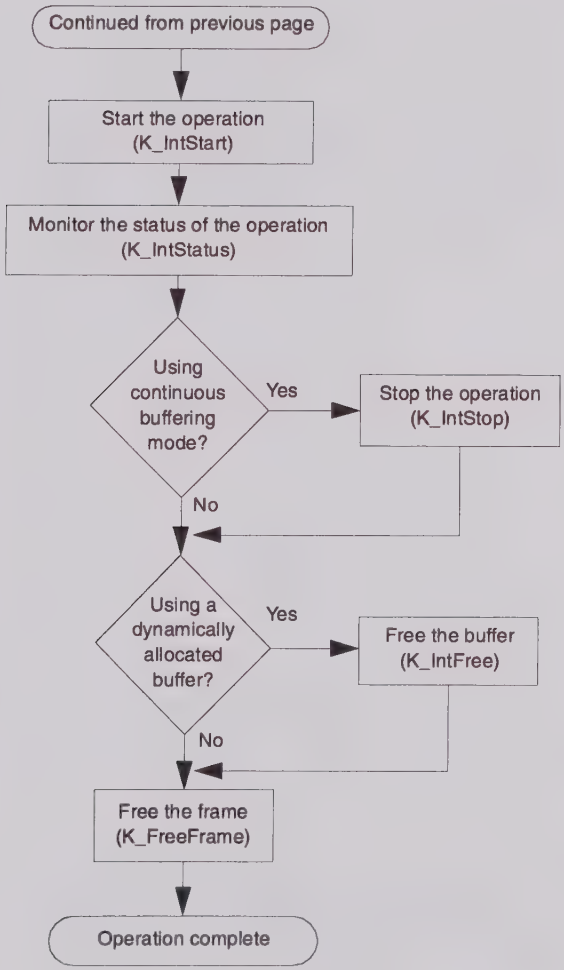
Steps for an Analog Input Operation (cont.)



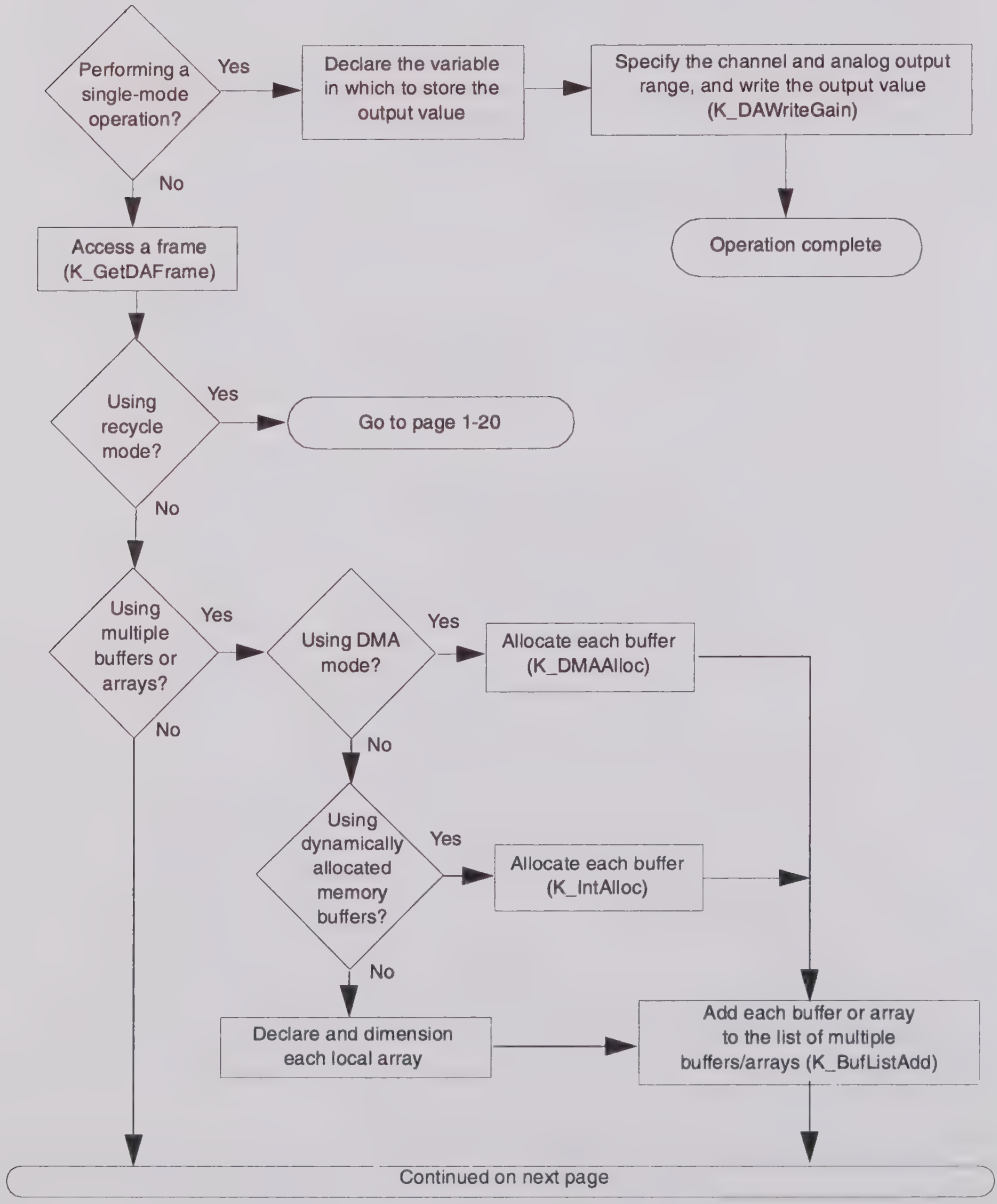
Steps for an Analog Output Operation (DAS-1700ST-DA Series, DAS-1702HR-DA Boards)


Performing a single-mode operation?
Yes
No
Declare the variable in which to store the output value
Specify the channel and write the output value (K_DAWriteGain)
Operation complete
Access a frame (K_GetDAFrame)
Using a dynamically allocated memory buffer?
Yes
No
Allocate a buffer (K_IntAlloc)
Specify the starting address of the buffer (K_SetBuf)
Declare and dimension a local array
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Continued on next page

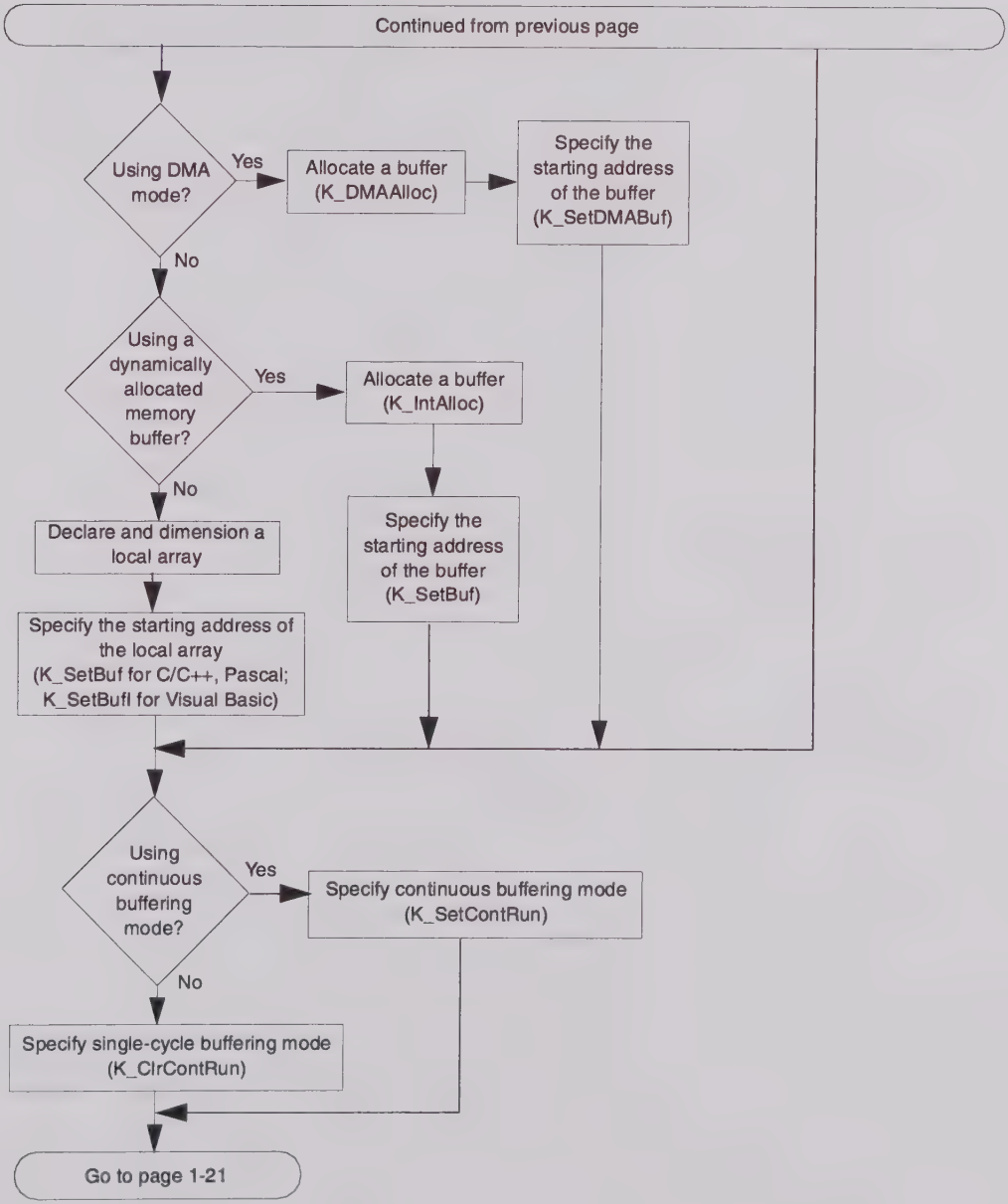

Continued from previous page
Writing to a group of channels?
Yes
Specify the first and last channels (K_SetStartStopChn)
No
Specify a single channel (K_SetChn)
Set the clock rate (K_SetClkRate)
Using continuous buffering mode?
Yes
Specify continuous buffering mode (K_SetContRun)
No
Specify single-cycle buffering mode (K_ClrContRun)
Using Visual Basic?
Yes
Load output values into program's local array
No
Using a dynamically allocated buffer?
Yes
Transfer data to the dynamically allocated buffer (K_MoveArrayToBuf)
Load output values into array or buffer
No
Continued on next page


Continued from previous page
Start the operation (K_IntStart)
Monitor the status of the operation (K_IntStatus)
Using continuous buffering mode?
Yes
Stop the operation (K_IntStop)
No
Using a dynamically allocated buffer?
Yes
Free the buffer (K_IntFree)
No
Free the frame (K_FreeFrame)
Operation complete

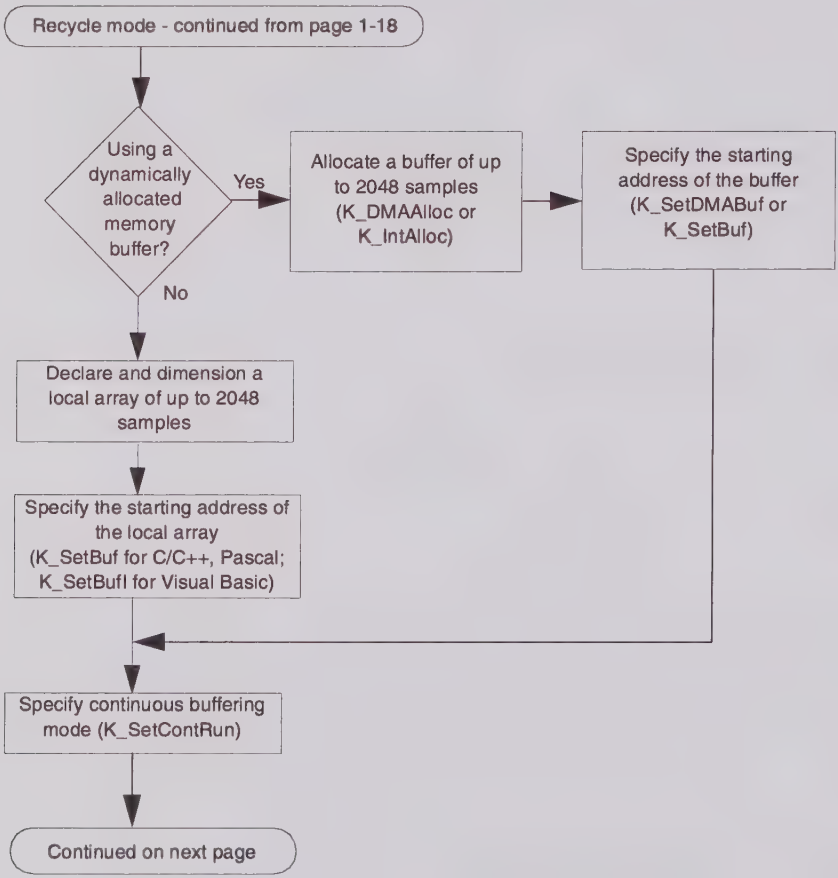
Steps for an Analog Output Operation (DAS-1700AO Series Boards)



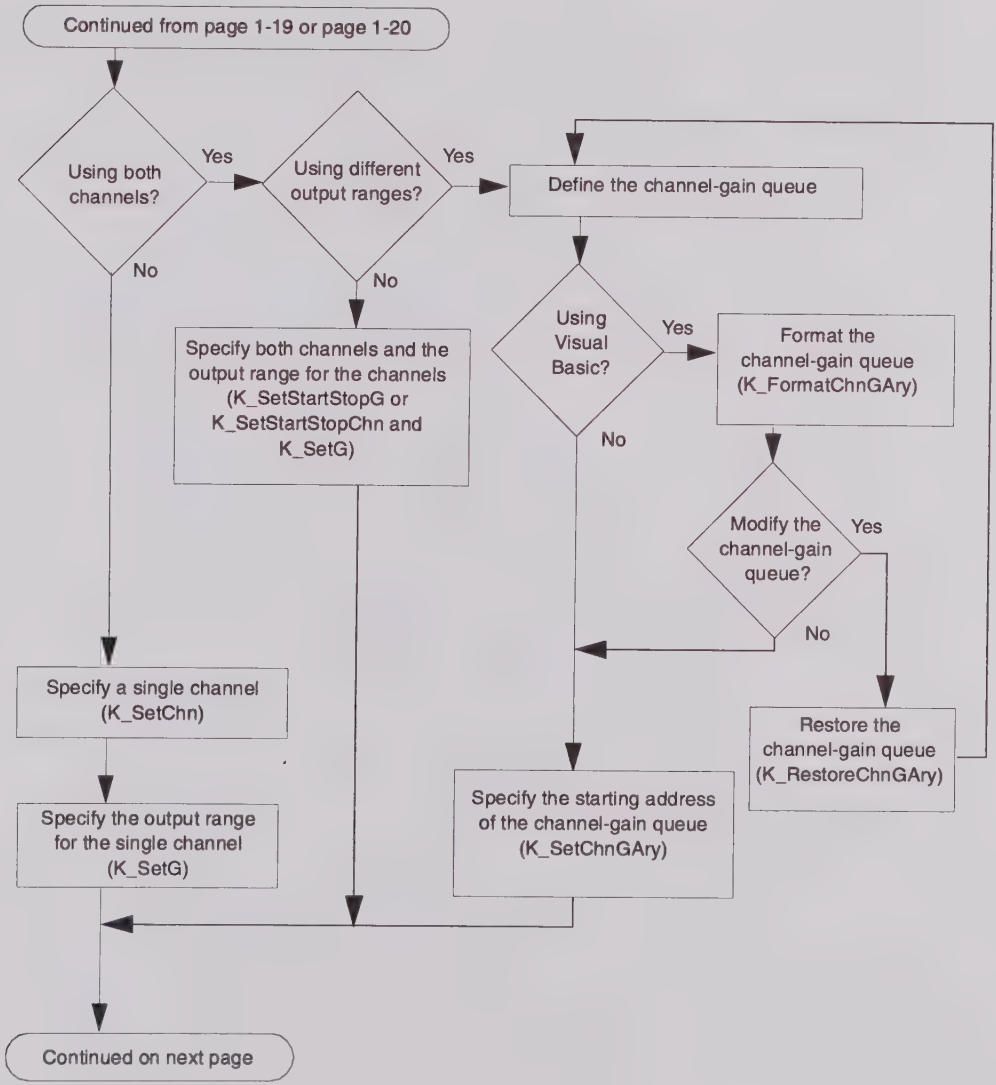
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)


Continued from previous page
Using DMA mode?
Yes
Allocate a buffer (K_DMAAlloc)
Specify the starting address of the buffer (K_SetDMABuf)
No
Using a dynamically allocated memory buffer?
Yes
Allocate a buffer (K_IntAlloc)
No
Declare and dimension a local array
Specify the starting address of the buffer (K_SetBuf)
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Using continuous buffering mode?
Yes
Specify continuous buffering mode (K_SetContRun)
No
Specify single-cycle buffering mode (K_ClrContRun)
Go to page 1-21

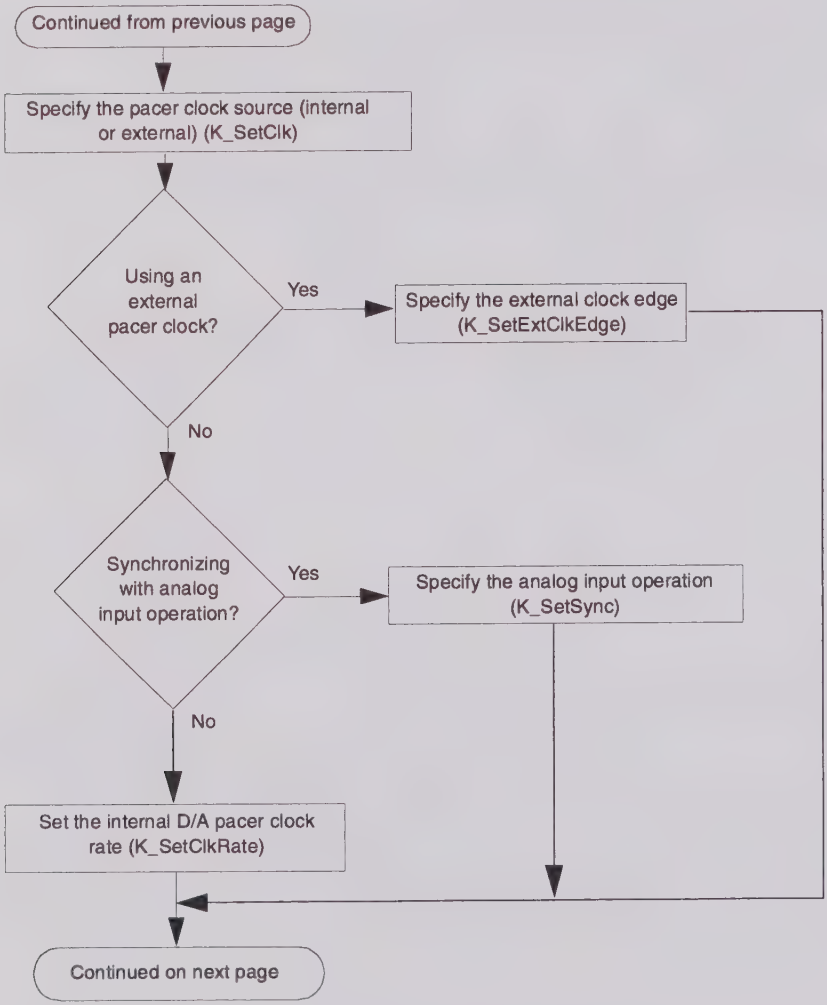
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)


Recycle mode - continued from page 1-18
Using a dynamically allocated memory buffer?
Yes
No
Allocate a buffer of up to 2048 samples (K_DMAAlloc or K_IntAlloc)
Specify the starting address of the buffer (K_SetDMABuf or K_SetBuf)
Declare and dimension a local array of up to 2048 samples
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Specify continuous buffering mode (K_SetContRun)
Continued on next page

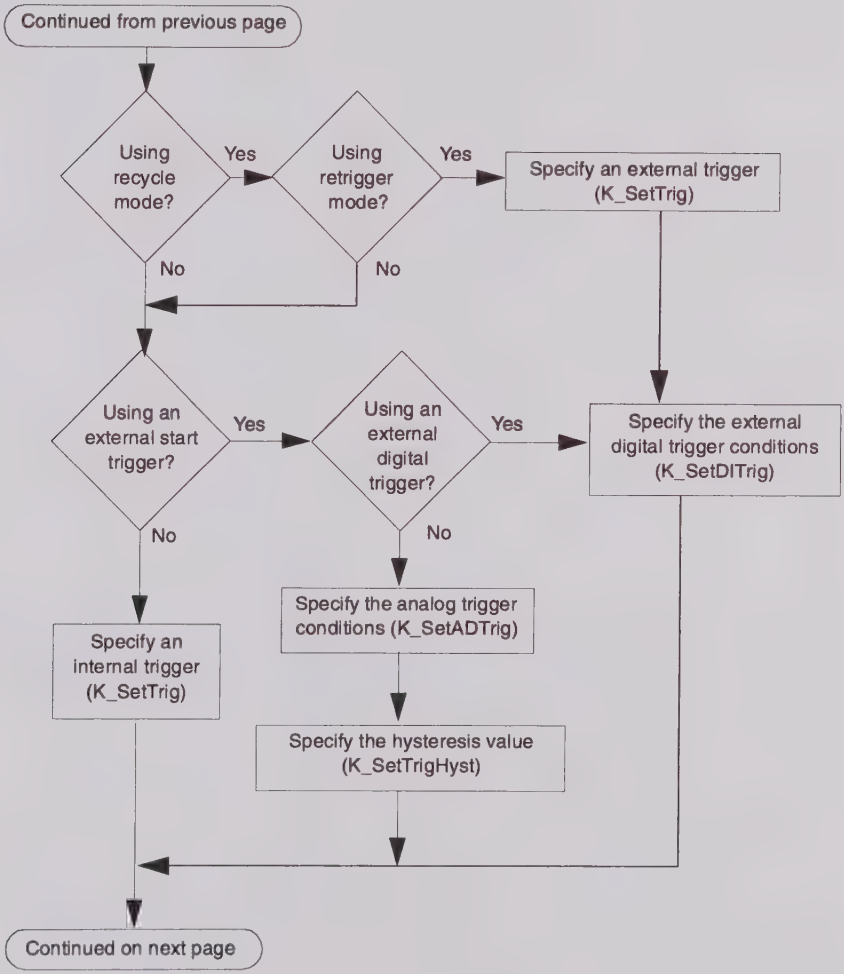
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)


Continued from page 1-19 or page 1-20
Using both channels?
Yes
No
Using different output ranges?
Yes
No
Define the channel-gain queue
Specify both channels and the output range for the channels (K_SetStartStopG or K_SetStartStopChn and K_SetG)
Using Visual Basic?
Yes
No
Format the channel-gain queue (K_FormatChnGAry)
Modify the channel-gain queue?
Yes
No
Specify a single channel (K_SetChn)
Specify the output range for the single channel (K_SetG)
Specify the starting address of the channel-gain queue (K_SetChnGAry)
Restore the channel-gain queue (K_RestoreChnGAry)
Continued on next page

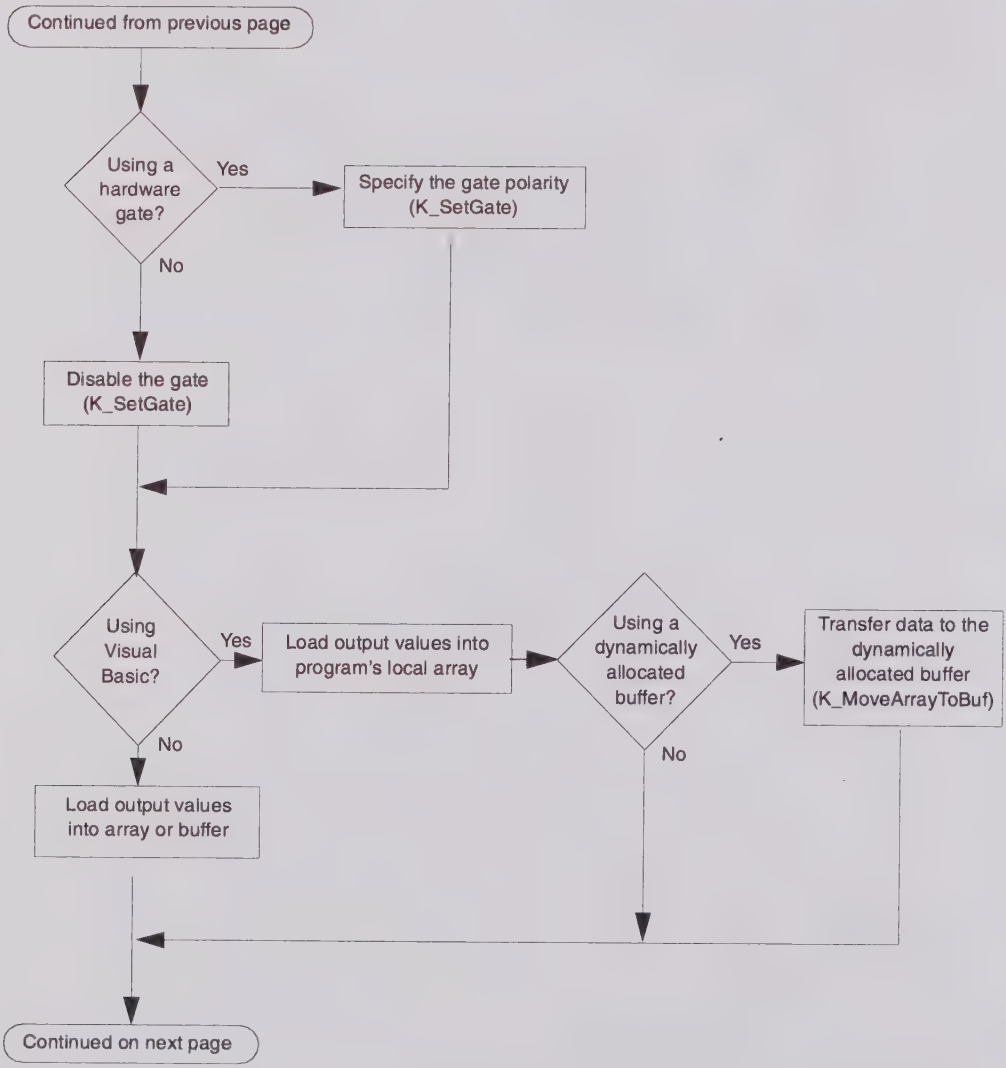
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)


Continued from previous page
Specify the pacer clock source (internal or external) (K_SetClk)
Using an external pacer clock?
Yes
Specify the external clock edge (K_SetExtClkEdge)
No
Synchronizing with analog input operation?
Yes
Specify the analog input operation (K_SetSync)
No
Set the internal D/A pacer clock rate (K_SetClkRate)
Continued on next page

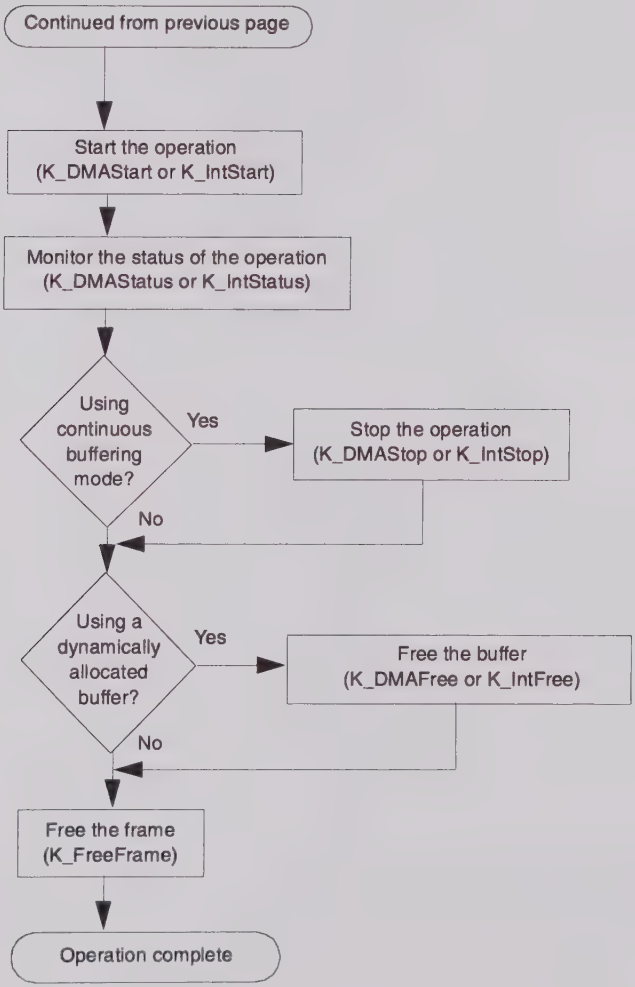
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)



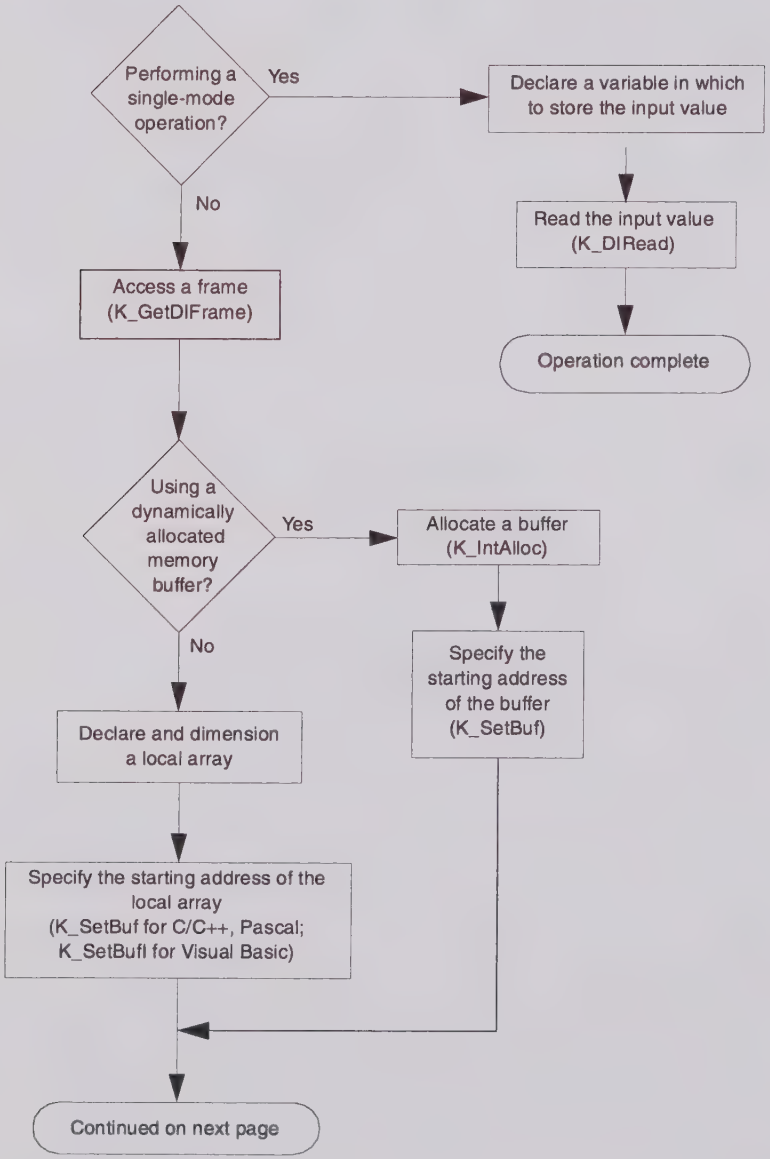
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)



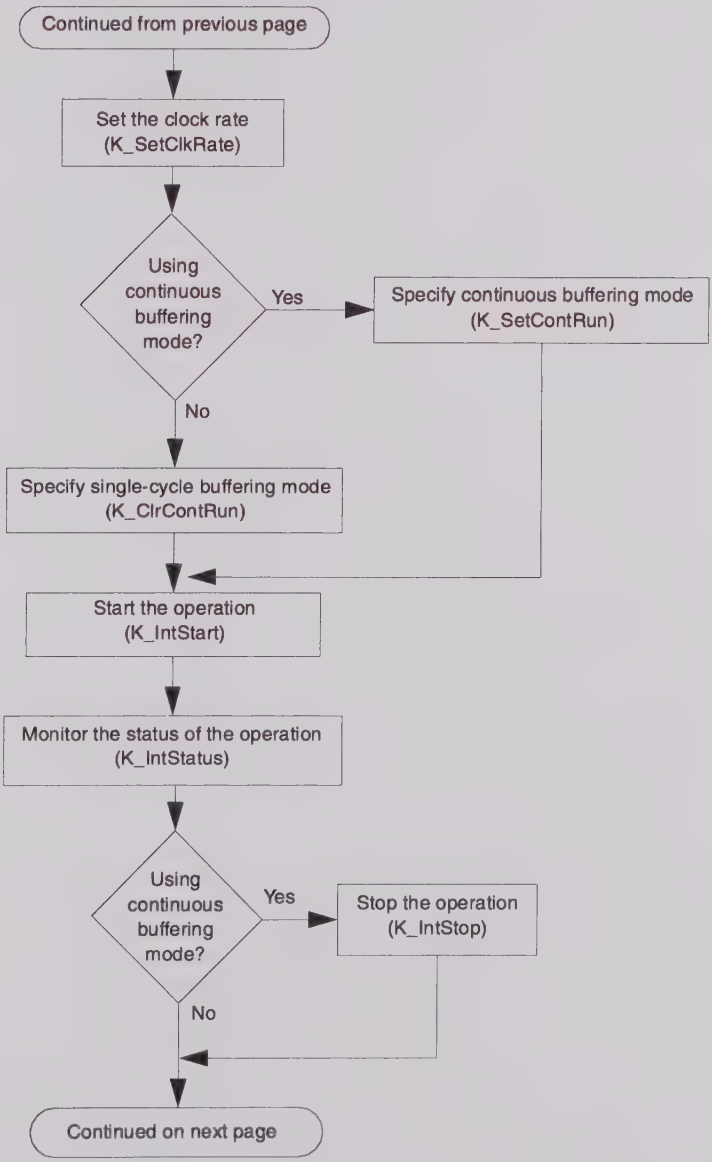
Steps for an Analog Output Operation (DAS-1700AO Series, cont.)



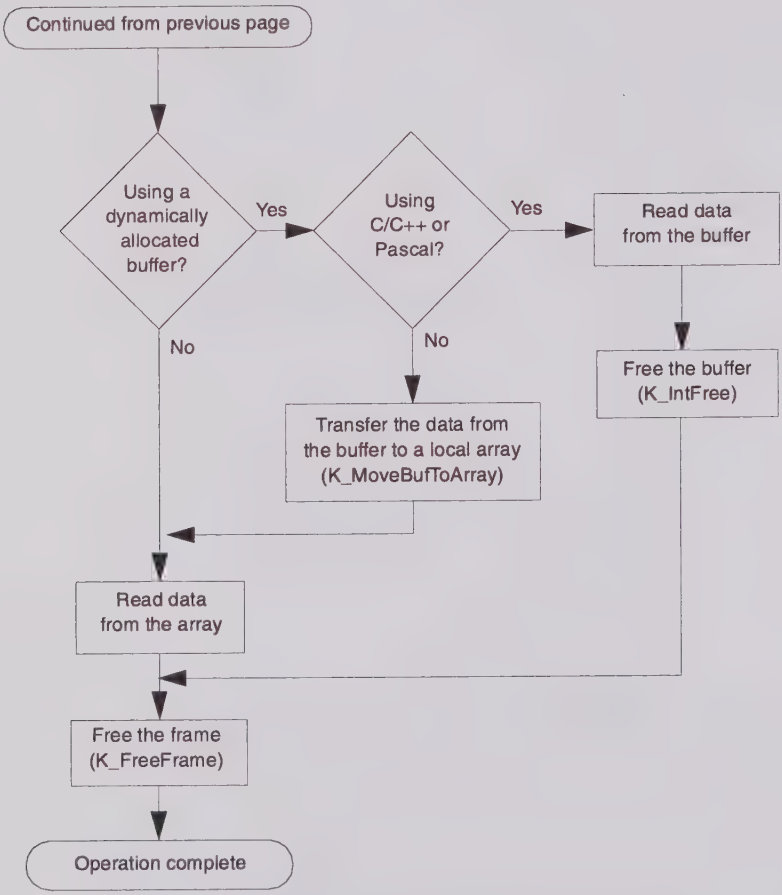
Steps for a Digital Input Operation


Performing a single-mode operation?
Yes
Declare a variable in which to store the input value
No
Read the input value (K_DIRead)
Access a frame (K_GetDIFrame)
Operation complete
Using a dynamically allocated memory buffer?
Yes
Allocate a buffer (K_IntAlloc)
No
Specify the starting address of the buffer (K_SetBuf)
Declare and dimension a local array
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Continued on next page

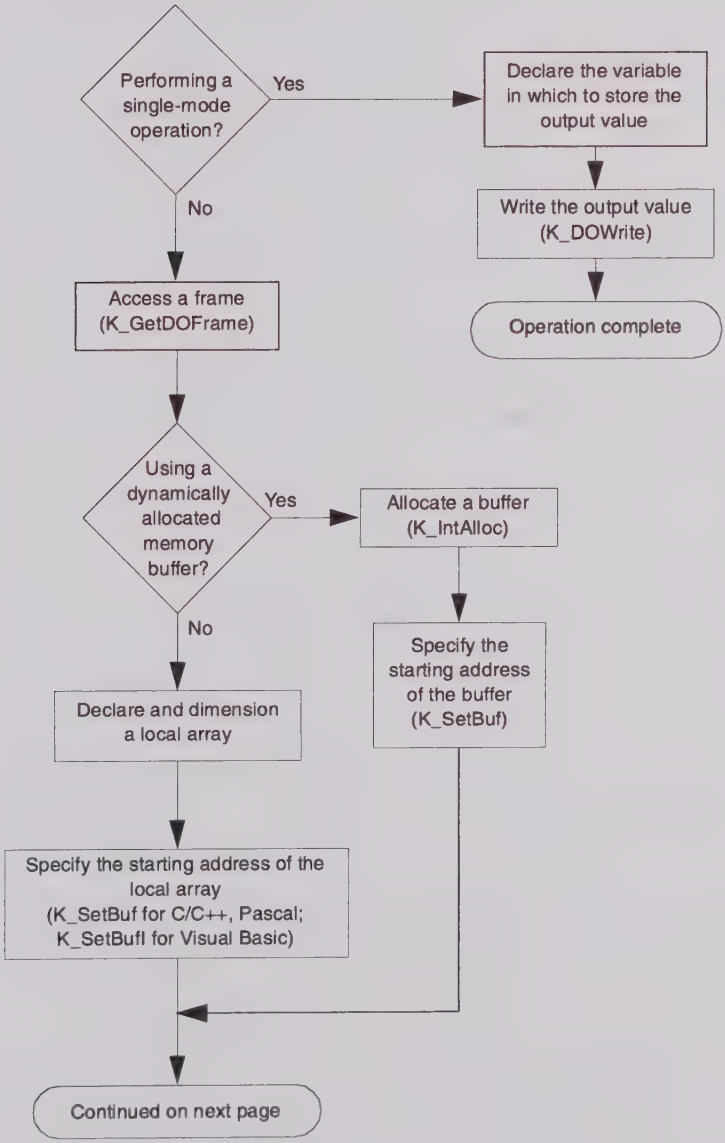
Steps for a Digital Input Operation (cont.)



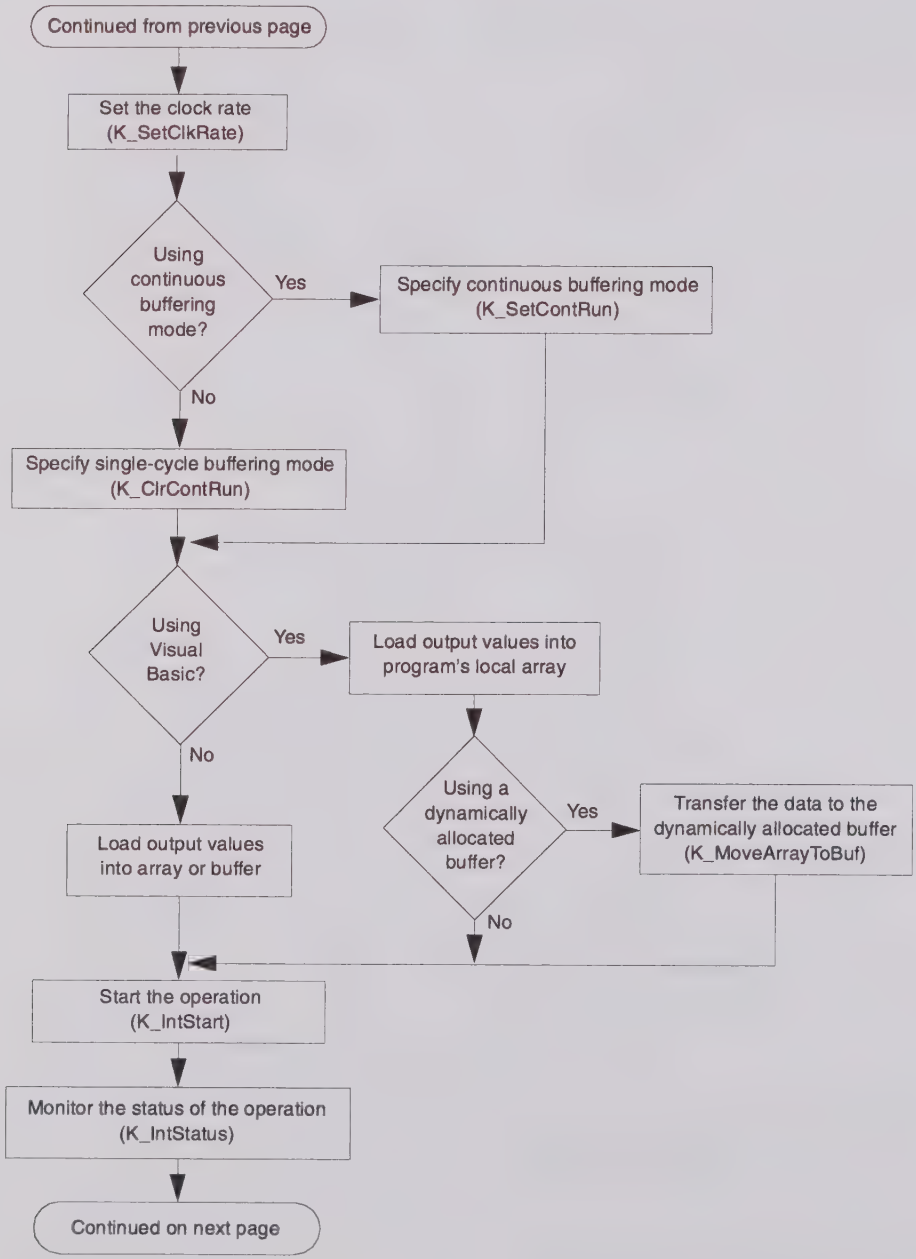
Steps for a Digital Input Operation (cont.)

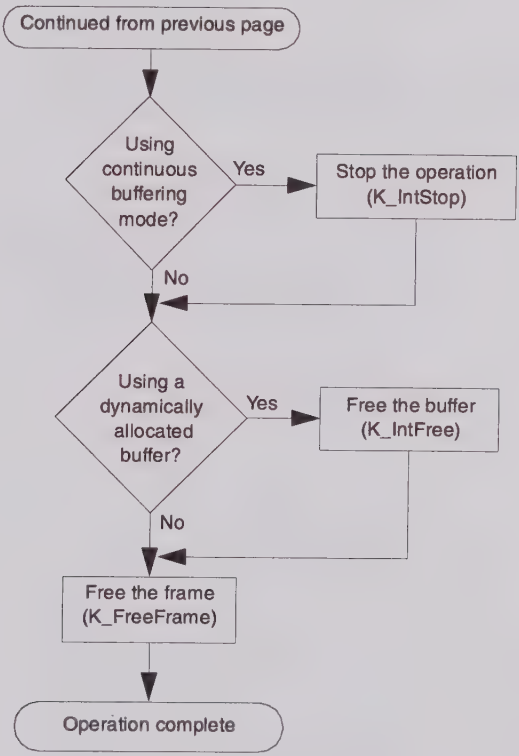


Steps for a Digital Output Operation


Performing a single-mode operation?
Yes
Declare the variable in which to store the output value
Write the output value (K_DOWrite)
Operation complete
No
Access a frame (K_GetDOFrame)
Using a dynamically allocated memory buffer?
Yes
Allocate a buffer (K_IntAlloc)
Specify the starting address of the buffer (K_SetBuf)
No
Declare and dimension a local array
Specify the starting address of the local array (K_SetBuf for C/C++, Pascal; K_SetBufI for Visual Basic)
Continued on next page

Steps for a Digital Output Operation (cont.)





Getting Help

If you need help installing or using the DAS-1700 Series Function Call Driver, call your local sales office or call the following number for technical support:

(508) 880-3000

Monday - Friday, 8:00 A.M. - 6:00 P.M., Eastern Time

An applications engineer will help you diagnose and resolve your problem over the telephone.

Please make sure that you have the following information available before you call:

DAS-1700 Series board configuration	Model	_____
	Serial #	_____
	Revision code	_____
	Base address setting	_____
	Interrupt level setting	_____
	Number of channels	_____
	Input configuration	single-ended, differential
	Range type	unipolar, bipolar
	DMA channels	_____
	Number of SSH-8s	_____
	Number of EXPs	_____
Computer	Manufacturer	_____
	CPU type	_____
	Clock speed (MHz)	_____
	Amount of RAM	_____
	Video system	_____
	BIOS type	_____
Operating system	DOS version	_____
	Windows version	_____
Software package	Name	_____
	Serial #	_____
	Version	_____
	Invoice/Order #	_____
Compiler (if applicable)	Language	_____
	Manufacturer	_____
	Version	_____
Accessories	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____
	Type/Number	_____

2

Available Operations

This chapter contains the background information you need to use the FCD functions to perform operations on DAS-1700 Series boards.

This chapter contains the following sections:

- **System Operations** - information on initializing the Function Call Driver, initializing a board, retrieving revision levels, and handling errors.
- **Analog Input Operations** - information on operation modes, frames, memory allocation and management, gains and ranges, channels, conversion modes, pacer clocks, buffering modes, and triggers.
- **Analog Output Operations (ST-DA, HR-DA)** - information on operation modes, frames, memory allocation and management, channels, and buffering modes.
- **Analog Output Operations (AO)** - information on operation modes, frames, memory allocation and management, channels, output ranges, pacer clocks, buffering modes, and triggers.
- **Digital I/O Operations** - information on operation modes, frames, memory allocation and management, the digital I/O channel, pacer clocks, and buffering modes.

System Operations

This section describes the miscellaneous and general maintenance operations that apply to DAS-1700 Series boards and to the DAS-1700 Series Function Call Driver. It includes information on initializing a driver, initializing a board, retrieving revision levels, and handling errors.

Initializing the Driver

You must initialize the DAS-1700 Series Function Call Driver and any other Keithley DAS Function Call Drivers you are using in your program. To initialize the drivers, use the **K_OpenDriver** function. You specify the driver you are using (DAS1700) and the configuration file that defines the use of the driver. This configuration file should already be defined, using the board's configuration utility (CFG1700.EXE). The driver returns a unique identifier for the driver; this identifier is called the driver handle.

You can specify a maximum of 30 driver handles for all the Keithley MetraByte drivers initialized from all your programs. If you no longer require a driver and you want to free some memory, or if you used all 30 driver handles, use the **K_CloseDriver** function to free a driver handle and close the associated driver. This function also closes all device and frame handles associated with the driver handle.

If the driver handle you free is the last driver handle specified for a Function Call Driver, the driver is shut down, and the DLLs associated with the driver are shut down and unloaded from memory.

Initializing a Board

The DAS-1700 Series Function Call Driver supports up to three boards. To specify the boards you want to use, use the **K_GetDevHandle** function, and specify the driver handle and the board number (0, 1, or 2). **K_GetDevHandle** verifies that the board is present. The driver returns a unique identifier for each board; this identifier is called the *device handle*.

Device handles allow you to communicate with more than one board. You use the device handle returned by **K_GetDevHandle** in subsequent function calls related to the associated board.

You can specify a maximum of 30 device handles for all the Keithley MetraByte boards accessed from all your programs. If you are no longer using a board and you want to free some memory, or if you used all 30 device handles, use the **K_FreeDevHandle** function to free a device handle.

Use **K_GetDevHandle** the first time you initialize a board only. To reinitialize a board during an operation, use the **K_DASDevInit** function, specifying the device handle. (All operations on the board must be stopped before you can use **K_DASDevInit**.)

Retrieving Revision Levels

If you are using functions from different Keithley DAS Function Call Drivers in the same program, or if you are having problems with your program, you may want to verify which versions of the Function Call Driver, Keithley DAS Driver Specification, and Keithley DAS Shell are used by your board.

The **K_GetVer** function allows you to get both the revision number of the DAS-1700 Series Function Call Driver and the revision number of the Keithley DAS Driver Specification to which the driver conforms.

The **K_GetShellVer** function allows you to get the revision number of the Keithley DAS Shell. (The Keithley DAS Shell is a group of functions shared by all DAS boards.)

Handling Errors

Each FCD function returns a code indicating the status of the function. To ensure that your program runs successfully, Keithley MetraByte recommends that you check the returned code after the execution of each function. If the status code equals 0, the function executed successfully and your program can proceed. If the status code does not equal 0, an error occurred; ensure that your program takes the appropriate action. Refer to Appendix A for a complete list of error causes and solutions. Each supported programming language uses a different procedure for error checking. Refer to the following pages of this guide for information:

C/C++	page 3-8
Pascal	page 3-16
Visual Basic for Windows	page 3-25

For C-language programs only, the DAS-1700 Series Function Call Driver provides the **K_GetErrMsg** function, which gets the address of the string corresponding to an error code.

Analog Input Operations

You can perform analog input operations on all DAS-1700 Series boards.

This section describes the following:

- Analog input operation modes available.
- How to access a frame.
- How to allocate and manage memory for analog input operations.
- How to specify channels and gains, a conversion mode, a clock source, a buffering mode, a trigger source, and a hardware gate for analog input operations.

Operation Modes

The operation mode determines which attributes you can specify for an analog input operation and how data is transferred from the board to computer memory. You can perform analog input operations in single mode, interrupt mode, or DMA (Direct Memory Access) mode, as described in the following sections.

Single Mode

In single mode, the board acquires a single sample from an analog input channel. The driver initiates the conversion; you cannot perform any other operation until the single-mode operation is complete.

Use the **K_ADRead** function to perform an analog input operation in single mode. You specify the device handle for the board you want to use, the analog input channel, the gain at which you want to read the signal, and the variable in which to store the converted data.

The data in the variable is a count value. Refer to Appendix B for information on converting a count value to voltage.

Interrupt Mode

In interrupt mode, the board acquires a single sample or multiple samples from one or more analog input channels. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your program. The hardware temporarily stores the acquired data in the onboard A/D FIFO (first-in, first-out data buffer for analog-to-digital conversions), and then transfers the data to a user-defined buffer in computer memory using an interrupt service routine.

Use the **K_IntStart** function to start an analog input operation in interrupt mode.

You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-23 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

The data in the user-defined buffer is stored as count values. Refer to Appendix B for information on converting the count values to voltages.

DMA Mode

DMA mode provides the fastest data transfer rates. In DMA mode, the board acquires a single sample or multiple samples from one or more analog input channels. A hardware clock initiates conversions. Once the analog input operation begins, control returns to your program. The hardware temporarily stores the acquired data in the onboard A/D FIFO and then transfers the data to a user-defined DMA buffer in the computer.

Note: You can perform an analog input operation in single-DMA mode or dual-DMA mode, depending on whether you specified one or two DMA channels in your configuration file. Refer to the *DAS-1700 Series User's Guide* for more information.

Use the **K_DMAStart** function to start an analog input operation in DMA mode.

You can specify either single-cycle or continuous buffering mode for DMA-mode operations. Refer to page 2-23 for more information on buffering modes. Use the **K_DMAStop** function to stop a continuous-mode DMA operation. Use the **K_DMAStatus** function to determine the current status of a DMA operation.

The data in the user-defined buffer is stored as count values. Refer to Appendix B for information on converting the count values to voltages.

Frames

Interrupt-mode and DMA-mode analog input operations require the use of frames. A frame is a data structure whose elements define the attributes of the operation. Use the **K_GetADFrame** function to access an analog input frame, called an A/D frame. The driver returns a frame handle, which is a unique identifier for the frame.

You specify the attributes of the operation using individual setup functions to define each element of the A/D frame. Use the frame handle returned by the driver in each setup function to ensure that you always define the same operation. For example, assume that you access an A/D frame with the frame handle `ADFrame`. To specify the channel on which to perform the operation, use the **K_SetChn** setup function, referencing the frame handle `ADFrame`. To specify the gain at which to read the channel, use the **K_SetG** setup function and reference the frame handle `ADFrame`.

When you are ready to perform the operation you set up, use the **K_IntStart** or **K_DMAStart** function to start the operation, again referencing the appropriate frame handle. Figure 2-1 illustrates the use of an A/D frame for a DMA-mode operation where the frame handle is `ADFrame`.

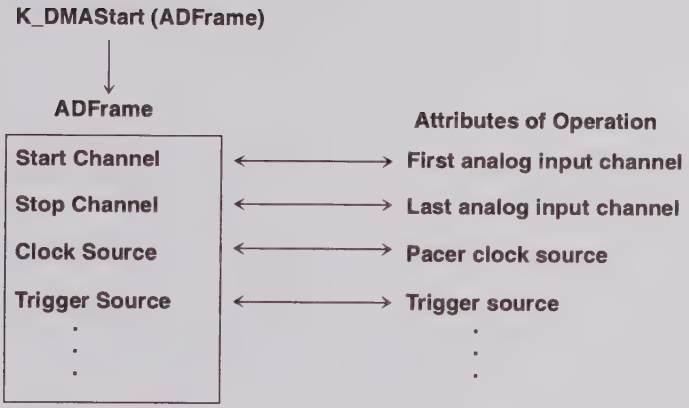


Figure 2-1. Frame-Based Operation

Frames help you create structured programs. They are useful for operations that have many defining attributes, since providing a separate argument for each attribute could make a function's argument list unmanageably long. In addition, some attributes, such as the clock source and trigger source, are only available for operations that use frames.

When you access a frame, the elements of that frame are set to their default values. You can use the **K_ClearFrame** function to reset all the elements of a frame to their default values.

If you want to perform an interrupt-mode or a DMA-mode operation and all frames been accessed, use the **K_FreeFrame** function to free a frame that is no longer in use. The frame elements remain as defined for the previous operation. You can redefine the frame elements as needed for your operation.

Table 2-1 lists the elements of an A/D frame, the default value of each element, the setup functions used to define each element, and the pages in this guide on which to find additional information.

Table 2-1. A/D Frame Elements

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
		K_SetDMABuf	page 4-129
		K_BufListAdd	page 4-9
Number of Samples	0	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
		K_SetDMABuf	page 4-129
		K_BufListAdd	page 4-9
Buffering Mode	Single-cycle	K_SetContRun	page 4-125
		K_ClrContRun[2]	page 4-22
Start Channel	0	K_SetChn	page 4-115
		K_SetStartStopChn	page 4-139
		K_SetStartStopG	page 4-141
Stop Channel	0	K_SetStartStopChn	page 4-139
		K_SetStartStopG	page 4-141
Gain	0 (gain of 1)	K_SetG	page 4-133
		K_SetStartStopG	page 4-141
Channel-Gain Queue	0 (NULL)	K_SetChnGAry	page 4-117
Conversion Mode	Paced	K_SetADFreeRun	page 4-102
		K_ClrADFreeRun[2]	page 4-20
SSH Mode	Disabled	K_SetSSH	page 4-137
Clock Source	Internal (A/D)	K_SetClk	page 4-120
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-123

Table 2-1. A/D Frame Elements (cont.)

Element	Default Value	Setup Function	Page Number
External Clock Edge	Negative	K_SetExtClkEdge	page 4-131
Burst Clock Rate[3]	6 (166,666.667 samples/s)[3]	K_SetBurstTicks	page 4-113
Trigger Source	Internal	K_SetTrig	page 4-146
Trigger Type	Digital	K_SetADTrig	page 4-106
		K_SetDITrig	page 4-127
Trigger Channel	0 (for analog trigger)	K_SetADTrig	page 4-106
	0 (channel 0, bit 0) (for digital trigger)	Not applicable[4]	Not applicable[4]
Trigger Polarity	Positive (for analog trigger)	K_SetADTrig	page 4-106
	Positive (for digital trigger)	K_SetDITrig	page 4-127
Trigger Sensitivity	Edge (for analog and digital trigger)	Not applicable[4]	Not applicable[4]
Trigger Level	0	K_SetADTrig	page 4-106
Trigger Hysteresis	0	K_SetTrigHyst	page 4-148
About-Trigger Mode	Disabled	K_SetAboutTrig	page 4-96
		K_ClrAboutTrig[2]	page 4-18
Hardware Gate	Disabled	K_SetGate	page 4-135

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default rate depends on the type of board specified in the configuration file and is 6 clock ticks (166,666.667 samples/second) for a DAS-1700ST Series board or DAS-1700AO Series board, or 20 clock ticks (50,000 samples/second) for a DAS-1700HR Series board.

[4] The default value of this element cannot be changed.

Memory Allocation and Management

Interrupt-mode and DMA-mode analog input operations require memory in which to store the acquired data. You can allocate a single memory buffer or array, or you can allocate multiple memory buffers or arrays to increase the number of samples you can acquire. The following sections describe the ways you allocate and manage memory.

Dimensioning Local Arrays

The simplest way to reserve memory is to dimension arrays within your program's memory area. You can dimension an unlimited number of local arrays. The advantage of this method is that the arrays are directly accessible to your program. The limitations of this method are as follows:

- Certain programming languages limit the size of local arrays.
- Local arrays may not be suitable for DMA-mode operations.
- Local arrays occupy permanent memory areas that cannot be freed to make them available to other programs or processes.
- You cannot use local arrays with Windows 95, 32-bit programs.

Since the DAS-1700 Series Function Call Driver stores data in 16-bit integers, you must dimension all local arrays as integers.

Dynamically Allocating Memory Buffers

The recommended way to reserve memory is to dynamically allocate memory buffers (up to 150) outside of your program's memory area. The advantages of this method are as follows:

- The number of buffers and the size of each buffer are limited by the amount of free physical memory available in your computer at run time.
- You can free dynamically allocated memory buffers to make them available to other programs or processes.

The limitation of this method is that, for Visual Basic for Windows, data in a dynamically allocated memory buffer is not directly accessible to your program. (For C/C++ and Pascal, the data is directly accessible to your program.) For Visual Basic, you must use the **K_MoveBufToArray**

function to move the data from the dynamically allocated buffer to the program's local array. (Refer to page 4-91 for more information.)

Use the **K_IntAlloc** function to dynamically allocate a memory buffer for interrupt-mode operations and the **K_DMAAlloc** function to dynamically allocate a memory buffer for DMA-mode operations. Specify the operation requiring the buffer and the number of samples to store in the buffer (up to 5,000,000 for **K_IntAlloc** and up to 65,536 for **K_DMAAlloc**). The driver returns the starting address of the buffer and a unique identifier for the buffer, called the memory handle.

When you no longer need the buffer, free it for another use by specifying this memory handle in the **K_IntFree** function (for interrupt-mode operations) or the **K_DMAFree** function (for DMA-mode operations).

Notes: If you are writing Windows 95, 32-bit programs, you must install and use the Keithley Memory Manager to allocate memory for data acquisition programs. If you are writing 16-bit Windows programs, it is strongly recommended that you install and use the Keithley Memory Manager. Refer to the *DAS-1700 Series User's Guide* for information.

For interrupt-mode operations, acquiring data into small memory buffers (typically fewer than 1,024 samples) can cause an A/D FIFO overflow error. The overflow error occurs because interrupts are processed more often (whenever the A/D FIFO is not empty). To optimize interrupt-mode operations, make sure that the size of your buffer is a multiple of 512 samples. If your application requires small memory buffers and high throughput, it is recommended that you use DMA mode - especially if you are using a single, small buffer in continuous mode.

The area used for dynamically allocated memory buffers is referred to as the *global heap*, which is an area of memory left unoccupied as your program and other programs run. If you have not installed the Keithley Memory Manager, the **K_IntAlloc** and **K_DMAAlloc** functions call the **GlobalAlloc** API function to allocate the desired buffer size from the global heap.

Dynamically allocated memory is guaranteed to be fixed and locked in memory. To eliminate page wrap conditions and to guarantee that dynamically allocated memory is suitable for use by the computer's 8237 DMA controller, **K_DMAAlloc** can allocate an area twice as large as

actually needed. After processing and/or saving this buffer's data elsewhere, use **K_DMAFree** to free the memory for other uses.

Assigning the Starting Addresses

After you dimension your arrays or allocate your buffers, you must assign the starting addresses of the arrays or buffers and the number of samples to store in the arrays or buffers.

Each supported programming language requires a particular procedure for dimensioning local arrays, allocating memory buffers, and assigning starting addresses. Refer to the following pages of this guide for information:

C/C++	page 3-3
Pascal	page 3-11
Visual Basic for Windows	page 3-17

If you are using multiple buffers or arrays, use the **K_BufListAdd** function to add each buffer or array to the list of multiple buffers or arrays associated with each operation and to assign the starting address of each buffer or array. Use the **K_BufListReset** function to clear all buffers/arrays from the list of multiple buffers/arrays.

Gains and Ranges

Each channel on a DAS-1700 Series board can measure analog input signals in one of four, software-selectable unipolar or bipolar analog input ranges. If you use EXP-1800 accessories with a DAS-1700 Series board, four more input ranges are available. The input range type (unipolar or bipolar) is initially set according to your configuration file; use **K_SetADMode** to reset the input range type. Refer to the *DAS-1700 Series User's Guide* for more information.

Table 2-2 lists the analog input ranges supported by DAS-1700 Series boards, and the gain and gain code associated with each range. Gain codes are used by the FCD functions to represent the gain.

Table 2-2. Analog Input Ranges

Boards	Location of Channel	Analog Input Range		Gain	Gain Code
		Bipolar	Unipolar		
DAS-1701ST DAS-1701ST-DA DAS-1701AO	Either onboard or on an EXP-1800 using a gain of 1	±5 V	0 to 5 V	1	0
		±1 V	0 to 1 V	5	1
		±0.1 V	0 to 0.1 V	50	2
		±0.02 V	0 to 0.02 V	250	3
	On an EXP-1800 using a gain of 50	±0.1 V	0 to 0.1 V	50	4
		±0.02 V	0 to 0.02 V	250	5
		±0.002 V	0 to 0.002 V	2500	6
		±0.0004 V	0 to 0.0004 V	12.5k	7
DAS-1702ST DAS-1702ST-DA DAS-1702HR[1] DAS-1702HR-DA[1] DAS-1702AO	Either onboard or on an EXP-1800 using a gain of 1	±10 V	0 to 10 V	1	0
		±5 V	0 to 5 V	2	1
		±2.5 V	0 to 2.5 V	4	2
		±1.25 V	0 to 1.25 V	8	3
	On an EXP-1800 using a gain of 50	±0.2 V	0 to 0.2 V	50	4
		±0.1 V	0 to 0.1 V	100	5
		±0.05 V	0 to 0.05 V	200	6
		±0.025 V	0 to 0.025 V	400	7

Notes

[1] Because the DAS-1700HR boards provide greater accuracy than the EXP-1800 boards, running the DAS-1702HR and DAS-1702HR-DA boards with an EXP-1800 that uses a gain of 50 may produce unsatisfactory results. Refer to the *DAS-1700 Series User's Guide* for more information.

For a single-mode operation, you specify the gain code in the **K_ADRead** function. For an interrupt-mode or DMA-mode operation, you specify the gain code in the **K_SetG** or **K_SetStartStopG** function.

The function you use depends on how you specify the channels, as described in the following section.

Channels

DAS-1700 Series boards are configured with either 16 single-ended or eight differential analog input channels. If you require additional channels on DAS-1700 Series boards, you can use up to 16 EXP-1800 expansion accessories or MB02 backplanes to increase the number of channels to a maximum of 256.

The input channel configuration (either single-ended or differential) is initially set according to the configuration file; use the **K_SetADConfig** function to reset the input channel configuration. Use **K_SetADCommonMode** to set the common-mode ground reference for DAS-1700 Series boards in a single-ended input configuration.

You can perform an analog input operation on a single channel or on a group of multiple channels. The following sections describe how to specify the channels you are using.

Specifying Channels when Using Expansion Accessories

To increase the number of channels, you can assign EXP-1800 expansion accessories or MB02 backplanes to consecutive onboard analog input channels, beginning with onboard channel 0. Note that onboard analog input channels assigned to EXP-1800 expansion accessories or MB02 backplanes must use a single-ended input configuration. You can also use the remaining onboard channels. Refer to the *DAS-1700 Series User's Guide* for information on connecting EXP-1800 expansion accessories or MB02 backplanes to DAS-1700 Series boards.

The analog input channel connections on an EXP-1800 expansion accessory are labelled from 0 to 15. These numbers are the *physical channel numbers*. If a system includes a DAS-1700 Series board and one or more EXP-1800 expansion accessories or MB02 backplanes, then that system contains duplicate physical channel numbers. To uniquely identify a physical channel, the Function Call Driver uses a scheme of *logical channel numbers*. The channel number argument for any FCD function must be specified as a logical channel number.

To determine the logical channel number corresponding to a particular physical channel number, use one of the following equations:

Physical channel is on a DAS-1700 Series board

$$LogicalChan\# = PhysicalChan\# + (15 \times NumEXPs)$$

Physical channel is on an EXP-1800 or MB02

$$LogicalChan\# = PhysicalChan\# + (16 \times EXP\#)$$

where

NumEXPs is an integer from 0 to 16 that identifies the number of EXP-1800 expansion accessories or MB02 backplanes connected to the DAS-1700 Series board, and

EXP# is an integer from 0 to 15 that indicates on which EXP-1800 expansion accessory the physical channel is located. (0 indicates the EXP-1800 board that has its output channel jumper set to 0.)

For example, consider the system illustrated in Figure 2-2, in which three EXP-1800 expansion accessories are connected to a DAS-1701ST.

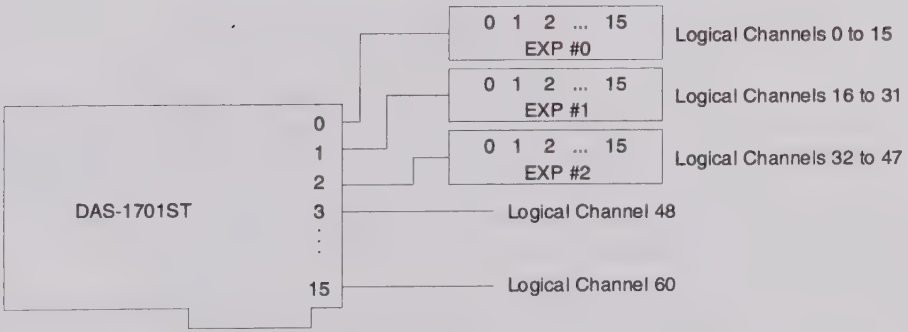


Figure 2-2. Example of Logical Channel Assignments

The logical channel that identifies channel 3 on the DAS-1701ST board is determined as follows:

$$LogicalChan\# = 3 + (15 \times 3) = 3 + 45 = 48$$

The logical channel that identifies channel 15 on the third EXP-1800 expansion accessory is determined as follows:

$$LogicalChan\# = 15 + (16 \times 2) = 15 + 32 = 47$$

Specifying a Single Channel

For single-mode analog input operations, you can acquire a single sample from a single analog input channel. Use the **K_ADRead** function to specify the channel and the gain code.

For interrupt-mode and DMA-mode analog input operations, you can acquire a single sample or multiple samples from a single analog input channel. Use the **K_SetChn** function to specify the channel and the **K_SetG** function to specify the gain code for the channel.

Refer to Table 2-2 on page 2-13 for a list of the analog input ranges supported by DAS-1700 Series boards and the gain code associated with each range.

Specifying a Group of Consecutive Channels

For interrupt-mode and DMA-mode analog input operations, you can acquire samples from a group of consecutive channels. Use the **K_SetStartStopChn** function to specify the first and last channels in the group of consecutive channels. The channels are sampled in order from first to last, and are then sampled again until the required number of samples is read.

For example, assume that the start channel is 4, the stop channel is 7, and you want to acquire a total of five samples. Your program reads a sample first from channel 4, then from channels 5, 6, and 7, and finally from channel 4 again.

You can specify a start channel that is higher than the stop channel. For example, assume that you are using a single-ended input configuration on a DAS-1701ST board and the start channel is 15, the stop channel is 2, and you want to acquire a total of five samples. Your program reads a sample first from channel 15, then from channels 0, 1, and 2, and finally from channel 15 again.

Use the **K_SetG** function to specify the gain code for all channels in the group. (All channels must use the same gain code.) Use the **K_SetStartStopG** function to specify the gain code, the start channel, and the stop channel in a single function call.

Refer to Table 2-2 on page 2-13 for a list of the analog input ranges supported by DAS-1700 Series boards and the gain code associated with each range.

Specifying Channels in a Channel-Gain Queue

For interrupt-mode and DMA-mode analog input operations, you can acquire samples from channels in a hardware channel-gain queue. In the channel-gain queue, you specify the channels you want to sample, the order in which you want to sample them, and a gain code for each channel.

You can set up the channels in a channel-gain queue either in consecutive order or in nonconsecutive order. You can also specify the same channel more than once, up to a total of 256 channels.

The channels are sampled in order from the first channel in the queue to the last channel in the queue. The channels in the queue are then sampled again until the specified number of samples is read.

Refer to Table 2-2 on page 2-13 for a list of the analog input ranges supported by DAS-1700 Series boards and the gain code associated with each range.

The way that you specify the channels and gains in a channel-gain queue depends on the language you are using. Refer to the following pages for information:

C/C++	page 3-7
Pascal	page 3-15
Visual Basic for Windows	page 3-23

After you create the channel-gain queue in your program, use the **K_SetChnGAry** function to specify the starting address of the channel-gain queue.

Conversion Modes

The conversion mode determines how the board regulates the timing of conversions when you are acquiring multiple samples from a single channel or from a group of multiple channels (known as a scan). For interrupt-mode and DMA-mode analog input operations, you can specify paced mode, burst mode, or burst mode with SSH (simultaneous sample-and-hold, as described in the following sections.

Refer to the *DAS-1700 Series User's Guide* for more information about conversion modes.

Paced Mode

Use paced mode if you want to accurately control the period between conversions of individual channels in a scan. Paced mode is the default conversion mode. To reset the conversion mode to paced mode, use the **K_ClrADFreeRun** function.

Burst Mode

Use burst mode if you want to accurately control both the period between conversions of individual channels in a scan and the period between conversions of the entire scan. Use the **K_SetADFreeRun** function to specify burst mode.

Burst Mode with SSH

Use burst mode with SSH (simultaneous sample-and-hold) if you want to accurately control both the period between conversions of individual channels in a scan and the period between conversions of the entire scan, and if you want to sample all the channels in a scan simultaneously. Use the **K_SetSSH** function to specify burst mode with SSH.

Notes: If you are using an SSH-8 accessory to simultaneously sample all channels in a scan, one extra tick of the burst mode conversion clock is required to allow the SSH-8 to sample and hold the values. Refer to the SSH-8 documentation for more information.

You cannot use an SSH-8 accessory with a DAS-1700HR Series board.

Clock Sources

DAS-1700 Series boards provide two clock sources for analog input operations: a pacer clock and a burst mode conversion clock. Each clock has a dedicated use. When performing interrupt-mode and DMA-mode analog input operations in paced mode, you use only the pacer clock; when performing interrupt-mode and DMA-mode analog input operations in burst mode and burst mode with SSH, you use both the pacer clock and the burst mode conversion clock. These clock sources are described in the following sections.

Pacer Clock

In paced mode, the pacer clock determines the period between the conversion of one channel and the conversion of the next channel. In burst mode and burst mode with SSH, the pacer clock determines the period between the conversions of one scan and the conversions of the next scan.

Note: The rate at which the computer can reliably read data from the board depends on a number of factors, including your computer, the operating system/environment, the gains of the channels, and other software issues.

Refer to the *DAS-1700 Series User's Guide* for more information about the pacer clock.

The DAS-1700 Series boards support two types of pacer clocks for analog input conversions: the internal A/D pacer clock and an external pacer clock. The following sections describe these two clock sources.

Internal A/D Pacer Clock

The internal A/D pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters are normally in idle states. When you start the analog input operation (using **K_IntStart** or **K_DMAStart**), a conversion is initiated. Note that a slight time delay occurs between the time the operation is started and the time the conversion is initiated.

After the first conversion is initiated, the counters are loaded with a count value and begin counting down. When the counters count down to 0, another conversion is initiated and the process repeats.

Because the counters use a 5 MHz time base, each count represents 0.2 µs. Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between conversions. For example, if you specify a count of 87654, the period between conversions is 17.53 ms (57 samples/s).

The following table shows the valid ranges of counts (clock ticks) you can specify, the corresponding periods between conversions, and the corresponding internal A/D pacer clock rates for the DAS-1700 Series boards. As you can see in the table, the lowest count value and smallest period between conversions correspond to the highest clock rate; the highest count value and longest period between conversions correspond to the lowest clock rate.

Board Series	Counts (Clock Ticks)	Periods between Conversions	Internal A/D Pacer Clock Rates (samples/second)
DAS-1700ST/AO	30 to 4,294,836,225	6 µs to 14.3 minutes	166,666.667 to 0.0012
DAS-1700HR	100 to 4,294,836,225	50 µs to 14.3 minutes	50,000 to 0.0012

When using the internal A/D pacer clock, use the following formula to determine the number of counts to specify:

$$\text{counts} = \frac{\text{5 MHz time base}}{\text{conversion rate}}$$

For example, if you want a conversion rate of 10 ksamples/s, specify a count of 500, as shown in the following equation:

$$\frac{5,000,000}{10,000} = 500$$

The conversion rate is the rate at which the analog-to-digital converter (ADC) initiates conversions; it does not take into account the number of channels you are using. For example, if you are using five channels and want a conversion rate of 1 ksamples/second per channel, specify 1,000 clock ticks, as shown in the following equation:

$$\left(\frac{5,000,000}{1,000}\right) \div 5 = 1,000$$

If the hardware cannot convert the analog input channels at the exact rate determined by the count you specify, the driver calculates a rate that is as close as possible to the count you specify. To determine the actual count used by the internal A/D pacer clock, use the **K_GetClkRate** function after you start the analog input operation. (Make sure that the frame specified in **K_GetClkRate** is an A/D frame.) Refer to page 4-62 for more information.

The internal A/D pacer clock is the default pacer clock. To reset the pacer clock source to the internal A/D pacer clock, use the **K_SetClk** function.

External Pacer Clock

You connect an external pacer clock to the XPCLK pin (pin 44) on the main I/O connector of DAS-1700 Series boards. When you start an analog input operation (using **K_IntStart** or **K_DMAStart**), conversions are armed. At the next active edge of the external pacer clock (and at every subsequent active edge of the external pacer clock), a conversion is initiated.

Use the **K_SetClk** function to specify an external pacer clock. Then, use the **K_SetExtClkEdge** function to specify the active edge (rising or falling) of the external pacer clock. A falling edge is the default active edge for the external pacer clock.

If you are using an external pacer clock, make sure that the clock initiates conversions at a rate that the analog-to-digital converter (ADC) can handle. (Refer to the *DAS-1700 Series User's Guide* for more information about ADC rates.)

Note: If you are using a DAS-1700AO Series board, you can use the same external pacer clock to pace both an analog input operation and an analog output operation.

Burst Mode Conversion Clock

In burst mode and burst mode with SSH, the burst mode conversion clock determines the period between the conversion of one channel in a scan and the conversion of the next channel in the scan.

Because the burst mode conversion clock uses a 1 MHz time base, each clock tick represents 1 µs. Use the **K_SetBurstTicks** function to specify the number of clock ticks between conversions. For example, if you specify 30 clock ticks, the period between conversions is 30 µs (33.33 ksamples/s).

The following table shows the valid ranges of counts (clock ticks) you can specify, corresponding periods between conversions, and corresponding burst mode conversion clock rates for the DAS-1700 Series boards. As with the internal A/D pacer clock, the lowest count value and smallest period between conversions correspond to the highest burst mode conversion clock rate; the highest count value and longest period between conversions correspond to the lowest burst mode conversion clock rate.

Board Series	Counts (Clock Ticks)	Periods between Conversions	Burst Rates (samples/second)
DAS-1700ST/AO	6 to 64	6 µs to 64 µs	166,666.667 to 15,625
DAS-1700HR	20 to 64	20 µs to 64 µs	50,000 to 15,625

When using the burst mode conversion clock, use the following formula to determine the number of clock ticks to specify:

$$\text{clock ticks} = \frac{\text{1 MHz time base}}{\text{burst mode conversion rate}}$$

For example, if you want a burst mode conversion rate of 20 ksamples/s, specify 50 clock ticks, as shown in the following equation:

$$\frac{1,000,000}{20,000} = 50$$

The driver checks the maximum burst mode conversion rate for the board you are using. If you specify a rate that is too fast for the board, the driver uses the maximum rate instead. To determine the actual number of clock ticks used by the burst mode conversion clock, use the **K_GetBurstTicks** function after you start the analog input operation. Refer to page 4-60 for more information.

Refer to the *DAS-1700 Series User's Guide* for more information about the burst mode conversion clock.

Buffering Modes

The buffering mode determines how the driver stores the converted data in the buffer. For interrupt-mode and DMA-mode analog input operations, you can specify either single-cycle buffering mode or continuous buffering mode.

Single-Cycle Mode

In single-cycle mode, after the board converts the specified number of samples and stores them in the buffer, the operation stops automatically. Single-cycle mode is the default buffering mode. To reset the buffering mode to single-cycle mode, use the **K_ClrContRun** function.

Continuous Mode

In continuous mode, the board continuously converts samples and stores them in the buffer until it receives a stop function; any values already stored in the buffer are overwritten as new data is converted and stored. Use the **K_SetContRun** function to specify continuous buffering mode.

Triggers

A trigger is an event that occurs based on a specified set of conditions. An interrupt-mode or DMA-mode analog input operation must have a start trigger to determine when acquisitions start. In addition, a DMA-mode analog input operation can use an optional second trigger, an about trigger, to determine when acquisitions stop.

The start trigger can be an internal trigger, an external analog trigger, or an external digital trigger. The about trigger, if used, is always an external digital trigger.

The following sections describe the supported trigger sources and the ways to acquire data using triggers.

Trigger Sources

You can specify an internal trigger, external analog trigger, or external digital trigger as described in the following sections.

The trigger event is not significant until the operation the trigger governs is started (using **K_DMAStart** or **K_IntStart**). The point at which conversions begin relative to the trigger event depends on the pacer clock. Refer to page 2-19 for more information about the pacer clock.

Internal Trigger

An internal trigger is a software trigger. The trigger event occurs when you start the operation. Note that there is a slight delay between the time you start the operation and the time the trigger event occurs. The internal trigger is the default trigger source. To reset the trigger source to an internal trigger, use the **K_SetTrig** function.

External Analog Trigger

You can use the signal on any analog input channel as the trigger signal for an external analog trigger. The trigger conditions for analog triggers are illustrated in Figure 2-3 and described as follows:

- **Positive-edge trigger** - The trigger event occurs the first time the trigger signal (analog input signal) changes from a voltage that is less than the specified trigger level to a voltage that is greater than the specified trigger level.
- **Negative-edge trigger** - The trigger event occurs the first time the trigger signal (analog input signal) changes from a voltage that is greater than the specified trigger level to a voltage that is less than the specified trigger level.

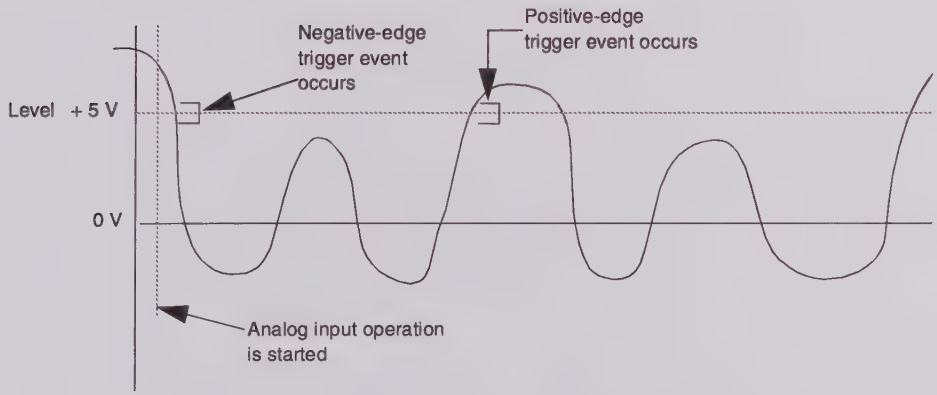


Figure 2-3. Analog Trigger Conditions

To specify an external trigger, use the **K_SetTrig** function. Then, use the **K_SetADTrig** function to specify the following:

- **Analog input channel to use as the trigger channel** - The trigger channel always measures signals at a gain of 1.
- **Trigger polarity and sensitivity** - The trigger can be a positive-edge or negative-edge.

- **Trigger level** - You specify the trigger level as a count value. The following table shows the valid values for the DAS-1700 Series boards.

Applicable Boards	Bipolar Range	Unipolar Range
DAS-1700ST/AO Series	–2048 to 2047	0 to 4095
DAS-1700HR Series	–32768 to 32767	0 to 65535

  For a bipolar input range type, the lowest count value (–2048 or –32768) represents negative full scale and the highest count value (2048 or 32767) represents positive full scale. For a unipolar input range type, 0 represents 0 V and the highest count value (4095 or 65535) represents positive full scale. Refer to Appendix B for information on how to convert a voltage value to a count.

You can also specify a hysteresis value to prevent noise from triggering an operation. Use the **K_SetTrigHyst** function to specify the hysteresis value. The point at which the trigger event occurs is described as follows:

- **Positive-edge trigger** - the analog signal must be below the specified trigger level by at least the amount of the hysteresis value and then rise above the trigger level before the trigger event occurs.
- **Negative-edge trigger** - the analog signal must be above the specified trigger level by at least the amount of the hysteresis value and then fall below the trigger level before the trigger event occurs.

The hysteresis value is an absolute number, which you specify as a count value from 0 to 4,095 for a DAS-1700ST/AO Series board, and from 0 to 65,535 for a DAS-1700HR Series board.

When you add the hysteresis value to the trigger level (for a negative-edge trigger) or subtract the hysteresis value from the trigger level (for a positive-edge trigger), the resulting value must be as follows:

- **DAS-1700ST/AO Series boards** - from 0 to 2,047 for a bipolar input range, or from 0 to 4,095 for a unipolar input range
- **DAS-1700HR Series boards** - from 0 to 32,767 for a bipolar input range, or from 0 to 65,535 for a unipolar input range

For example, assume that you are using a negative-edge trigger on a channel of a DAS-1701ST board configured for an analog input range of ±5 V. If the trigger level is +4.8 V (1966 counts), you can specify a hysteresis value of 0.1 V (41 counts) because 1966 + 41 is less than 2,047, but you cannot specify a hysteresis value of 0.3 V (123 counts) because 1966 + 123 is greater than 2,047. Refer to Appendix B for information on how to convert a voltage value to a count.

In Figure 2-4, the specified trigger level is +4 V and the hysteresis value is 0.1 V. The analog signal must be below +3.9 V and then rise above +4 V before a positive-edge trigger event occurs; the analog signal must be above +4.1 V and then fall below +4 V before a negative-edge trigger event occurs.

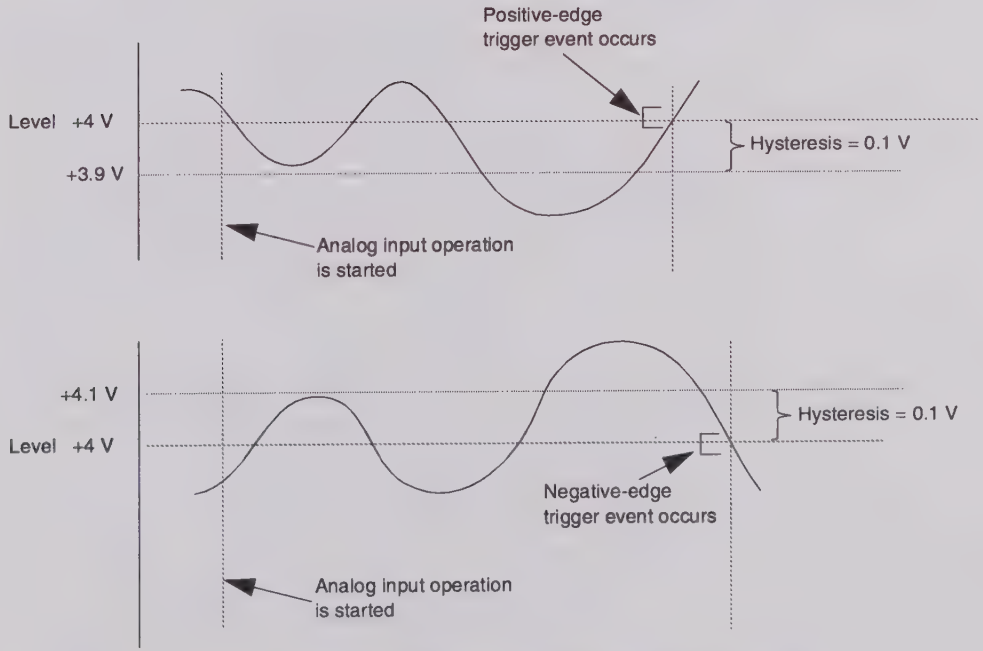


Figure 2-4. Using a Hysteresis Value

Note: The analog trigger is software-based trigger. When you start the analog input operation (using **K_IntStart** or **K_DMAStart**), the driver samples the specified channel until the trigger condition is met. Note that a slight delay occurs between the time the trigger condition is met and the time the driver realizes the trigger condition is met and begins conversions. Control does not return to your application until the trigger condition is met. (To terminate the operation if a trigger event does not occur, press **Ctrl+Break**.)

If you are programming in Visual Basic for Windows and you are using a DAS-1700HR Series board, you must represent a trigger level or hysteresis value greater than 32,767 as a negative integer. For more information, refer to page 3-24.

External Digital Trigger

An external digital trigger occurs when a rising edge or a falling edge is detected on the digital signal connected to the TGIN pin (pin 46) on the main I/O connector of DAS-1700 Series boards.

Use **K_SetTrig** to specify an external digital trigger source. Use **K_SetDITrig** to specify whether you want the trigger event to occur on a rising edge of the digital trigger signal (positive-edge trigger) or on a falling edge of the digital trigger signal (negative-edge trigger). The digital trigger conditions are illustrated in Figure 2-5.

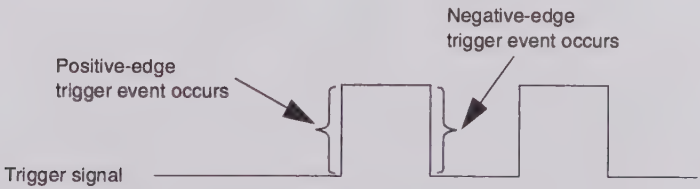


Figure 2-5. Digital Trigger Conditions

The following subsection describe the ways you can acquire data using triggers.

Post-Trigger Acquisition

Use post-trigger acquisition in applications where you want to collect data after a specific trigger event. You specify a start trigger only; the start trigger determines when the operation starts and can be either an internal or external analog trigger, or an external digital trigger. The point at which the operation stops depends on the buffering mode.

To specify post-trigger acquisition, perform the following steps:

1. Specify the buffering mode, if necessary.
   - If you want acquisition to continue until you stop it using **K_DMAStop** or **K_IntStop**, use **K_SetContRun** to set the buffering mode to continuous.
   - If you want acquisition to stop after a specified number of samples is acquired, use **K_ClrContRun** to set the buffering mode to single-cycle. The operation stops as soon as the board acquires the number of samples specified by **K_SetBuf**, **K_SetDMABuf**, **K_SetBufI**, or **K_BufListAdd**.
2. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external trigger source (specify external for an analog or digital trigger).
   - If you specify an external start trigger in **K_SetTrig**, define the start trigger conditions using **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger).
3. If necessary, use **K_ClrAboutTrig** to disable the about trigger.

Pre-Trigger Acquisition

Use pre-trigger acquisition in applications where you want to collect data before a specific trigger event. You specify a start trigger and an about trigger. The start trigger determines when the operation starts and can be either an internal trigger, or an external analog or digital trigger. The about trigger is an external digital trigger; the operation stops when the about-trigger event occurs. Pre-trigger mode is available with DMA-mode operations only.

To specify pre-trigger acquisition, perform the following steps:

1. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external start trigger source (specify external for an analog or digital trigger).
   - If you specify an external start trigger in **K_SetTrig**, define the start trigger conditions using **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger).
2. Use **K_SetAboutTrig** to enable the about trigger and to set the number of samples to one.

Note: The minimum number of samples that you can specify in **K_SetAboutTrig** is one.

---

3. Specify the active edge for the about trigger, if necessary.
   - If the start trigger is an internal trigger or an external analog trigger, use **K_SetDITrig** to specify the active edge for the about trigger.
   - If the start trigger is an external digital trigger, the active edge used for the start trigger is also used for the about trigger. It is not necessary to use **K_SetDITrig** again.

About-Trigger Acquisition

Use about-trigger acquisition in applications where you want to collect data both before and after a specific trigger event. You specify a start trigger and an about trigger. The start trigger determines when the operation starts and can be either an internal trigger, or an external analog or digital trigger. The about trigger is an external digital trigger; the

operation stops after a specified number of samples has been acquired after the about-trigger event occurs. About-trigger acquisition is available with DMA-mode operations only.

To specify about-trigger acquisition, perform the following steps:

1. Specify the start trigger.
   - Use **K_SetTrig** to specify an internal or an external start trigger source (specify external for an analog or digital trigger).
   - If you specify an external start trigger in **K_SetTrig**, define the start trigger conditions using **K_SetADTrig** (for an analog trigger) or **K_SetDITrig** (for a digital trigger).
2. Use **K_SetAboutTrig** to enable the about trigger and to specify the desired number of post-trigger samples.
3. Specify the active edge for the about trigger, if necessary.
   - If the start trigger is an internal trigger or an external analog trigger, use **K_SetDITrig** to specify the active edge for the about trigger.
   - If the start trigger is an external digital trigger, the active edge used for the start trigger is also used for the about trigger. It is not necessary to use **K_SetDITrig** again.

Hardware Gates

A hardware gate is an externally applied digital signal that determines whether conversions occur. You connect the gate signal to the TGIN pin (pin 46) on the main I/O connector of DAS-1700 Series boards.

If you start an interrupt-mode or DMA-mode analog input operation (using **K_IntStart** or **K_DMAStart**) and the hardware gate is enabled, the state of the gate signal determines whether conversions occur, as follows:

- If you specify a positive gate, conversions occur only if the signal to TGIN is high; if the signal to TGIN is low, conversions are inhibited.
- If you specify a negative gate, conversions occur only if the signal to TGIN is low; if the signal to TGIN is high, conversions are inhibited.

Use the **K_SetGate** function to enable and disable the hardware gate and to specify the gate polarity (positive or negative). The default state of the hardware gate is disabled.

Note: You cannot use the hardware gate with an external digital trigger. If you use a digital trigger at one point in your program and later want to use a hardware gate, you must first disable the digital trigger. You disable the digital trigger by specifying an internal trigger in **K_SetTrig** or by setting up an analog trigger (using the **K_SetADTrig** function).

Analog Output Operations (ST-DA, HR-DA)

This section describes how to perform analog output operations on the DAS-1700ST-DA Series and DAS-1702HR-DA boards. You can also perform analog output operations on the DAS-1700AO Series boards. Because the DAS-1700AO Series boards provide additional features for analog output operations, a separate Analog Output Operations section exists for these boards, beginning on page 2-40.

This section describes the following:

- Analog output operation modes available.
- How to access a frame.
- How to allocate and manage memory for analog output operations.
- How to specify the following for an analog output operation: channels, a clock rate, and a buffering mode.

Note: You cannot use an external trigger or external pacer clock when performing an analog output operation on the DAS-1700ST-DA Series and the DAS-1702HR-DA board.

Operation Modes

The operation mode determines which attributes you can specify for an analog output operation. You can perform analog output operations in single mode or interrupt mode, as described in the following sections.

Single Mode

In single mode, the driver writes a single value to a single analog output channel and then updates the channel; you cannot perform any other operation until the single-mode operation is complete.

Use the **K_DAWriteGain** function to start an analog output operation in single mode. You specify the board you want to use, the analog output channel, and the value you want to write.

You specify the analog output value as a count value. Refer to Appendix B for information on converting a voltage value to a count.

Interrupt Mode

In interrupt mode, the driver writes a single value or multiple values to a single analog output channel or to a group of analog output channels simultaneously and then updates all the channels on the board. A hardware clock paces the analog output operation. Once the analog output operation begins, control returns to your program.

Use the **K_IntStart** function to start an analog output operation in interrupt mode.

You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-39 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

You specify the analog output values as count values. Refer to Appendix B for information on converting a voltage value to a count.

Frames

Interrupt-mode analog output operations on the DAS-1700ST-DA Series and DAS-1702HR-DA board require frames. Use the **K_GetDAFrame** function to access an analog output frame, called a D/A (digital-to-analog) frame. The driver returns the frame handle for the operation. Refer to page 2-6 for more information about frames.

Table 2-3 lists the elements of a D/A frame, the default value of each element, the setup functions used to define each element, and the pages in this guide on which to find additional information.

Table 2-3. D/A Frame Elements (DAS-1700ST-DA Series and DAS-1702HR-DA)

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
Number of Samples	0	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
Buffering Mode	Single-cycle	K_SetContRun	page 4-125
		K_ClrContRun[2]	page 4-22
Start Channel	0	K_SetChn	page 4-115
		K_SetStartStopChn	page 4-139
Stop Channel	0	K_SetStartStopChn	page 4-139
Clock Source	Internal	Not applicable[3]	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-123

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default value of this element cannot be changed.

Memory Allocation and Management

Interrupt-mode analog output operations use a single memory buffer to store the data to be written to the analog output channels.

Since an analog output operation typically requires a small array of data, you can reserve a memory buffer by dimensioning an array within your program's memory area. Because the DAS-1700 Series Function Call Driver writes data as 16-bits, you must dimension all local arrays as integers.

If you are using a group of analog output channels, when you start the analog output operation (using **K_IntStart**), the driver simultaneously writes one value to each channel in the group. The driver writes the first value in the array to the first channel, the second value in the array to the second channel, the third value in the array to the third channel, and so on.

To ensure predictable results, make sure that the number of values stored in the array is an even multiple of the number of channels in the group. For example, if you are using a group of four analog output channels on a DAS-1701ST-DA board, you can dimension an array of 100 values, but you should not dimension an array of 75 values.

After you dimension your array, you must assign the starting address of the array and the number of samples stored in the array.

Each supported programming language requires a particular procedure for dimensioning an array and assigning the starting address. Refer to the following pages in this guide for information:

C/C++	page 3-6
Pascal	page 3-14
Visual Basic for Windows	page 3-22

You can also use the **K_IntAlloc** function to dynamically allocate a memory buffer, if desired. You specify the operation requiring the buffer and the number of values you want to store in the buffer (up to a maximum of 5,000,000). The driver returns the starting address of the buffer and a memory handle for the buffer. (Memory handles are unique identifiers for buffers.) When you no longer need the buffer, you can free

it for another use by specifying this memory handle in the **K_IntFree** function.

For Visual Basic for Windows, data in a dynamically allocated memory buffer is not directly accessible to your program. You must use the **K_MoveArrayToBuf** function to move the data from the allocated buffer to the program's local array. Refer to page 4-89 for more information.

Note: You cannot use a local array with Windows 95, 32-bit programs; you must use **K_IntAlloc** to dynamically allocate a memory buffer. You must also install the Keithley Memory Manager. Refer to the *DAS-1700 Series User's Guide* for information.

Channels

DAS-1700ST-DA Series and DAS-1702HR-DA boards contain digital-to-analog converters (DACs), each of which is associated with an analog output channel. The DAS-1700ST-DA Series boards provide four analog output channels; the DAS-1702HR-DA board provides two analog output channels.

For single-mode analog output operations, the driver writes a single value to a single analog output channel and then updates the channel. Use the **K_DAWriteGain** function to specify the channel.

For interrupt-mode analog output operations, the options for specifying analog output channels are described as follows:

- To write a single value or multiple values to a single analog output channel, use the **K_SetChn** function to specify the channel. At each pulse of the pacer clock, the driver writes a new value to the specified channel.
- To write a single value or multiple values to a group of analog output channels simultaneously, use the **K_SetStartStopChn** function to specify the first and last channels in the group. At each pulse of the pacer clock, the driver writes new values to the channels in the group.

  For example, assume that you are using a DAS-1701ST-DA, the start channel is 0, the stop channel is 1, and your array contains two waveforms (0, 4095, 1, 4094, 2, 4093 . . . 4095, 0). At the first pulse

of the pacer clock, the driver writes 0 to channel 0 and 4095 to channel 1, at the next pulse of the pacer clock, the driver writes 1 to channel 0 and 4094 to channel 1, and so on.

Note: For analog output operations on the DAS-1700ST-DA Series and the DAS-1702HR-DA boards, the start channel number must be lower than or equal to the stop channel number.

The DAS-1700ST-DA Series and the DAS-1702HR-DA boards use an analog output range of ±10 V for the channels; you cannot change the output range.

Pacer Clock

When performing interrupt-mode analog output operations on the DAS-1700ST-DA Series and DAS-1702HR-DA board, you can use the internal A/D pacer clock to determine the update rate (how often the driver writes to the specified analog output channels and updates all the channels on the board).

Note: You can use the internal A/D pacer clock only if it is not being used by another operation.

The internal A/D pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters are normally in an idle state. When you start the analog output operation (using **K_IntStart**), the driver writes values from the array to the specified analog output channels and then updates all the channels on the board. Note that a slight time delay occurs between the time the operation is started and the time the channels are written to and updated.

The counters are loaded with a count value and begin counting down. When the counters count down to 0, the driver again writes to the specified channels and updates all the channels on the board. The process then repeats.

Because the counters use a 5 MHz time base, each count represents 0.2 µs. Use the **K_SetClkRate** function to specify the number of counts

(clock ticks) between updates. For example, if you specify a count of 5000, the period between updates is 1 ms (1 ksamples/s); if you specify a count of 87654, the period between updates is 17.53 ms (57 samples/s).

The following table shows the valid ranges of counts (clock ticks) between updates you can specify, the corresponding periods between updates, and the corresponding internal A/D pacer clock rates. The lowest count value and smallest period between updates correspond to the highest clock rate; the highest count value and longest period between updates correspond to the lowest clock rate.

Boards	Counts (Clock Ticks)	Periods between Updates	Internal A/D Pacer Clock Rates (samples/second)
DAS-1701ST-DA DAS-1702ST-DA	30 to 4,294,836,225	6 µs to 14.3 minutes	166,666.667 to 0.0012
DAS-1702HR-DA	100 to 4,294,836,225	50 µs to 14.3 minutes	50,000 to 0.0012

Note: The driver accepts a count value as low as 30 (DAS-1700ST-DA Series) or 100 (DAS-1702HR-DA board). However, since a FIFO is not used to buffer values for analog output operations on the DAS-1700ST-DA Series or DAS-1702HR-DA board, a low count value may cause an overrun error. The maximum observed update rate for the internal pacer clock is 1 ksamples/s when running under Windows 3.1 or Windows 3.11 for Workgroups. These rates would indicate a minimum count value of 5,000 when running under Windows 3.1 or Windows 3.11 for Workgroups.

Use the following formula to determine the number of counts to specify:

$$\text{counts} = \frac{\text{5 MHz time base}}{\text{update rate}}$$

For example, if you want to update the analog output channels at a rate of 500 samples/s, specify a count value of 10,000, as shown in the following equation:

$$\frac{5,000,000}{500} = 10,000$$

At each pulse of the pacer clock, the driver writes new values to those analog output channels specified in **K_SetChn** or **K_SetStartStopChn** and, at the same time, updates all analog output channels on the board with their current values. For example, assume that you are using a DAS-1702ST-DA board and specified start channel 0 and stop channel 1 in **K_SetStartStopChn**. At each pulse of the pacer clock, the driver writes new values to channels 0 and 1 and then updates all four channels on the board with their current values.

If the hardware cannot write to the analog output channel at the exact rate determined by the count you specify, the driver calculates a rate that is as close as possible to the count you specify. To determine the actual count used by the internal A/D pacer clock, use the **K_GetClkRate** function after you start the operation. (Make sure that the frame specified in **K_GetClkRate** is a D/A frame.) Refer to page 4-62 for more information.

Buffering Modes

The buffering mode determines how the driver writes the values in the buffer to the analog output channels. For interrupt-mode analog output operations, you can specify either single-cycle buffering mode or continuous buffering mode.

Single-Cycle Mode

In single-cycle mode, after the driver writes the values stored in the buffer, the operation stops automatically. Single-cycle mode is the default buffering mode. To reset the buffering mode to single-cycle mode, use the **K_ClrContRun** function.

Continuous Mode

In continuous mode, the driver continuously writes values from the buffer until the program issues a stop function; when all the values in the buffer have been written, the driver writes the values to the buffer again. Use the **K_SetContRun** function to specify continuous buffering mode.

Analog Output Operations (AO)

This section describes the following aspects of performing analog output operations on the DAS-1700AO Series boards:

- Analog output operation modes available.
- How to access a frame.
- How to allocate and manage memory for analog output operations.
- How to specify the following for an analog output operation: channels and output ranges, a clock source, a buffering mode, a trigger source, and a hardware gate.

Operation Modes

The operation mode determines which attributes you can specify for an analog output operation. For the DAS-1700AO Series boards, you can perform analog output operations in single mode, interrupt mode, DMA mode, or recycle mode, as described in the following sections.

Single Mode

In single mode, the driver writes a single value to a single analog output channel and then updates the channel; you cannot perform any other operation until the single-mode operation is complete.

Use the **K_DAWriteGain** function to start an analog output operation in single mode. You specify the board you want to use, the analog output channel, the gain code for the output range, and the value you want to write.

You specify the analog output value as a count value. Refer to Appendix B for information on converting a voltage value to a count.

Interrupt Mode

In interrupt mode, the driver transfers values from computer memory to the onboard D/A FIFO (first-in, first-out data buffer for digital-to-analog operations). A hardware pacer clock determines the rate at which the

board writes values from the onboard D/A FIFO to the analog output channels. The driver continues to transfer values from computer memory to the onboard D/A FIFO using an interrupt service routine. Once the analog output operation begins, control returns to your program.

Use the **K_IntStart** function to start an analog output operation in interrupt mode.

You can specify either single-cycle or continuous buffering mode for interrupt-mode operations. Refer to page 2-53 for more information on buffering modes for analog output operations. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

You specify the analog output values as count values. Refer to Appendix B for information on converting a voltage value to a count.

DMA Mode

In DMA mode, the driver transfers values from computer memory to the onboard D/A FIFO. A hardware pacer clock determines the rate at which the board writes values from the onboard D/A FIFO to the analog output channels. The driver continues to transfer values from computer memory to the onboard D/A FIFO using the DMA controller. Once the analog output operation begins, control returns to your program.

Note: Dual-DMA mode is not supported for analog output operations on the DAS-1700AO Series boards.

Use the **K_DMAStart** function to start an analog output operation in DMA mode.

You can specify either single-cycle or continuous buffering mode for DMA-mode operations. Refer to page 2-53 for more information on buffering modes. Use the **K_DMAStop** function to stop a continuous-mode DMA operation. Use the **K_DMAStatus** function to determine the current status of a DMA operation.

You specify the analog output values as count values. Refer to Appendix B for information on converting a voltage value to a count.

Recycle Mode

Recycle mode lets you generate the same waveform continuously. In recycle mode, the driver transfers a maximum of 2048 output values from computer memory to the onboard D/A FIFO. A hardware pacer clock determines the rate at which the board writes values from the onboard D/A FIFO to the analog output channels. The output values in the D/A FIFO are written continuously until the operation is stopped. Once the analog output operation begins, control returns to your program.

If the user-defined buffer in computer memory contains 2048 values or fewer and you specify continuous buffering mode, the driver assumes that you want to use recycle mode. The board automatically uses the onboard D/A FIFO instead of the computer's interrupt or DMA resources, providing the fastest data transfer rate (up to 200 ksamples/s).

Use the **K_DMAStart** or **K_IntStart** function to start an analog output operation in recycle mode. Use the **K_DMAStop** or **K_IntStop** function to stop the recycle-mode operation. Use the **K_DMAStatus** or **K_IntStatus** function to determine the current status of a recycle-mode operation.

Frames

Interrupt-mode, DMA-mode, and recycle-mode analog output operations require frames. Use the **K_GetDAFrame** function to access an analog output frame, called a D/A (digital-to-analog) frame. The driver returns the frame handle for the operation. Refer to page 2-6 for more information about frames.

Table 2-4 lists the elements of a D/A frame for the DAS-1700AO Series boards, the default value of each element, the setup functions used to define each element, and the pages in this guide on which to find additional information.

Table 2-4. D/A Frame Elements (DAS-1700AO Series)

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
		K_SetDMABuf	page 4-129
		K_BufListAdd	page 4-9
Number of Samples	0	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
		K_SetDMABuf	page 4-129
		K_BufListAdd	page 4-9
Buffering Mode	Single-cycle	K_SetContRun	page 4-125
		K_ClrContRun[2]	page 4-22
Start Channel	0	K_SetChn	page 4-115
		K_SetStartStopChn	page 4-139
		K_SetStartStopG	page 4-141
Stop Channel	0	K_SetStartStopChn	page 4-139
		K_SetStartStopG	page 4-141
Gain	0 (±5 V)	K_SetG	page 4-133
		K_SetStartStopG	page 4-141
Channel-Gain Queue	0 (NULL)	K_SetChnGAry	page 4-117
Clock Source	Internal D/A	K_SetClk	page 4-120
		K_SetSync	page 4-144
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-123
External Clock Edge	Negative	K_SetExtClkEdge	page 4-131

Table 2-4. D/A Frame Elements (DAS-1700AO Series) (cont.)

Element	Default Value	Setup Function	Page Number
Trigger Source	Internal	K_SetTrig	page 4-146
Trigger Type	Digital	K_SetADTrig	page 4-106
		K_SetDITrig	page 4-127
Trigger Channel	0 (for analog trigger)	K_SetADTrig	page 4-106
	0 (channel 0, bit 0) (for digital trigger)	Not applicable[3]	Not applicable[3]
Trigger Polarity	Positive (for analog trigger)	K_SetADTrig	page 4-106
	Positive (for digital trigger)	K_SetDITrig	page 4-127
Trigger Sensitivity	Edge (for analog and digital trigger)	Not applicable[3]	Not applicable[3]
Trigger Level	0	K_SetADTrig	page 4-106
Trigger Hysteresis	0	K_SetTrigHyst	page 4-148
About-Trigger Mode	Disabled	K_SetAboutTrig	page 4-96
		K_ClrAboutTrig[2]	page 4-18
Hardware Gate	Disabled	K_SetGate	page 4-135

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] You cannot change the default value of this element.

Memory Allocation and Management

Interrupt-mode, DMA-mode, and recycle-mode analog output operations require one or more memory buffers in which to store the values to be written to the analog output channels. (If you use multiple memory buffers for a recycle-mode operation, the total number of values in all the memory buffers cannot exceed 2048.)

The ways you can reserve memory are described in the following sections.

Dimensioning Local Arrays

You can dimension an unlimited number of local arrays within your program's memory area. The advantage of this method is that the arrays are directly accessible to your program. Because an analog output operation typically requires a small array of data, dimensioning a local array is the simplest way to reserve memory. However, note the following limitations of this method:

- Certain programming languages limit the size of local arrays.
- Local arrays may not be suitable for DMA-mode operations.
- Local arrays occupy permanent memory areas that cannot be freed to make them available to other programs or processes.
- You cannot use local arrays with Windows 95, 32-bit programs.

Since the DAS-1700 Series Function Call Driver stores data in 16-bit integers, you must dimension all local arrays as integers.

Dynamically Allocating Memory Buffers

Another way to reserve memory is to dynamically allocate memory buffers (up to 150) outside of your program's memory area. The advantages of this method are as follows:

- The number of buffers and the size of each buffer are limited by the amount of free physical memory available in your computer at run-time.
- You can free dynamically allocated memory buffers to make them available to other programs or processes.

The limitation of this method is that, for Visual Basic for Windows, the data in a dynamically allocated memory buffer is not directly accessible to your program. (For C/C++ and Pascal, the data is directly accessible to your program.) For Visual Basic for Windows, you must use the **K_MoveArrayToBuf** function to move the data from the program's local array to the dynamically allocated buffer. Refer to page 4-89 for more information.

When you dynamically allocate a memory buffer for an analog output operation on the DAS-1700AO Series boards, use the appropriate function to specify the operation requiring the buffer and the number of samples to store in the buffer, as follows:

If you are using	Use	To specify up to
Interrupt mode	**K_IntAlloc**	5,000,000 samples
DMA mode	**K_DMAAlloc**	65,536 samples
Recycle mode	**K_IntAlloc** or **K_DMAAlloc**	2048 samples

The driver returns the starting address of the buffer and a unique identifier for the buffer, called the memory handle.

When you no longer need the buffer, free it for another use by specifying this memory handle in the **K_IntFree** function (if you used **K_IntAlloc** to allocate the buffer) or the **K_DMAFree** function (if you used **K_DMAAlloc** to allocate the buffer).

Notes: If you are writing Windows 95, 32-bit programs, you must install and use the Keithley Memory Manager to allocate memory for data acquisition operations. If you are writing 16-bit programs, it is strongly recommended that you use the Keithley Memory Manager. Refer to the *DAS-1700 Series User's Guide* for information.

The area used for dynamically allocated memory buffers is referred to as the *global heap*, which is an area of memory left unoccupied as your program and other programs run. If you did not install the Keithley Memory Manager, the **K_IntAlloc** and **K_DMAAlloc** functions call the **GlobalAlloc** API function to allocate the desired buffer size from the global heap.

Dynamically allocated memory is guaranteed to be fixed and locked in memory. To eliminate page wrap conditions and to guarantee that dynamically allocated memory is suitable for use by the computer's 8237 DMA controller, **K_DMAAlloc** may allocate an area twice as large as actually needed. After processing and/or saving this buffer's data elsewhere, use **K_DMAFree** to free the memory for other uses.

Assigning the Starting Addresses

After you dimension your arrays or allocate your buffers, you must assign the starting addresses of the buffers or arrays and the number of samples to store in the buffers or arrays. If you are using recycle mode, make sure that the number of samples you specify for the operation does not exceed 2048.

Each supported programming language requires a particular procedure for assigning the starting addresses; refer to the following pages for more information:

C/C++	page 3-3
Pascal	page 3-11
Visual Basic for Windows	page 3-17

If you are using multiple buffers or arrays, use the **K_BufListAdd** function to add each buffer or array to the list of multiple buffers or arrays associated with each operation and to assign the starting address of each buffer or array. Use the **K_BufListReset** function to clear the list of multiple buffers/arrays.

Analog Output Ranges

You can write an analog output value to analog output channels on a DAS-1700AO Series board in one of two, software-selectable ranges. Table 2-5 lists the analog output ranges supported by DAS-1700AO Series boards and the gain code associated with each range. (The gain code is used by the FCD functions to represent the output range.)

Table 2-5. Analog Output Ranges (DAS-1700AO Series)

Range	Gain Code
±5 V	0
±10 V	1

For single-mode operations, specify the analog output range in the **K_DAWriteGain** function. For interrupt-mode, DMA-mode, or recycle-mode operations, specify the analog output range in the **K_SetG** or **K_SetStartStopG** function; the function you use depends on how you specify the channels, as described in the following section.

Channels

DAS-1700AO Series boards contain two digital-to-analog converters (DACs), each of which is associated with an analog output channel. The following sections describe how to specify the channels you are using.

Specifying a Single Channel

For single-mode analog output operations, the board writes a single value to a single analog output channel and then updates the channel. Use the **K_DAWriteGain** function to specify the channel and the output range.

For interrupt-mode, DMA-mode, and recycle-mode analog output operations, the board can write a single value or multiple values to a single analog output channel. Use the **K_SetChn** function to specify the channel and the **K_SetG** function to specify the output range. At each pulse of the pacer clock, the board writes a new value to the specified channel and then updates both channels with their current values.

Specifying Both Channels Using the Same Output Range

For interrupt-mode, DMA-mode, and recycle-mode analog output operations, the board can write a single value or multiple values to both analog output channels simultaneously when both channels use the same analog output range.

At each pulse of the pacer clock, the board writes two values, the first value to channel 0 and the next value to channel 1, and then updates both channels simultaneously. For example, assume that your memory buffer contains two waveforms (0, 4095, 1, 4094, 2, 4093 . . . 4095, 0). At the first pulse of the pacer clock, the board writes 0 to channel 0 and 4095 to channel 1 and then updates the channels; at the next pulse of the pacer clock, the board writes 1 to channel 0 and 4094 to channel 1 and then updates the channels.

Use the **K_SetStartStopChn** function to specify the start channel (channel 0) and stop channel (channel 1) and the **K_SetG** function to specify the output range. Use the **K_SetStartStopG** function to specify the start channel, the stop channel, and the output range in a single function call.

Refer to Table 2-5 on page 2-47 for a list of the analog output ranges supported by DAS-1700AO Series boards and the gain code associated with each output range.

Specifying Both Channels Using Different Output Ranges (Channel-Gain Queue)

For interrupt-mode, DMA-mode, and recycle-mode analog output operations, the board can write a single value or multiple values to both analog output channels simultaneously when each channel uses a different analog output range. The board uses a 2-entry channel-gain queue, with channel 0 and its output range as the first entry and channel 1 and its output range as the second entry.

At each pulse of the pacer clock, the board writes two values, the first value to channel 0 and the next value to channel 1, and then updates both channels simultaneously. For example, assume that your memory buffer contains two waveforms (0, 4095, 1, 4094, 2, 4093 . . . 4095, 0). At the first pulse of the pacer clock, the board writes 0 to channel 0 and 4095 to channel 1 and then updates the channels; at the next pulse of the pacer clock, the board writes 1 to channel 0 and 4094 to channel 1 and then updates the channels.

Refer to Table 2-5 on page 2-47 for a list of the analog output ranges supported by DAS-1700AO Series boards and the gain code associated with each output range.

The way that you specify the channels and output ranges in a channel-gain queue depends on the language you are using. Refer to the following for information:

C/C++	page 3-7
Pascal	page 3-15
Visual Basic for Windows	page 3-23

After you create the channel-gain queue in your program, use the **K_SetChnGAry** function to specify the starting address of the channel-gain queue.

Pacer Clock

When performing interrupt-mode, DMA-mode, and recycle-mode analog output operations, the pacer clock determines the update rate. The update rate is the rate at which the board writes new values from the D/A FIFO to the analog output channels and then updates both channels. You can specify the internal D/A pacer clock, an external pacer cock, or a clock source used for an analog input operation. The pacer clocks are described in the following sections; refer to the *DAS-1700 Series User's Guide* for more information.

Note: The rate at which the computer can reliably transfer data to the board depends on a number of factors, including your computer, the operation system/environment, the range of the channels, and other issues.

Internal D/A Pacer Clock

The internal D/A pacer clock uses an onboard 5 MHz time base. When you start the analog output operation (using **K_IntStart** or **K_DMAStart**), the driver sets up the hardware, transfers values to the onboard D/A FIFO, and then starts the internal D/A pacer clock. At the first pulse of the internal D/A pacer clock, the driver updates the analog output channels with values from the D/A FIFO. After a specified number of clock ticks, the driver updates the channels again and the process repeats.

Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between updates.

Because the internal D/A pacer clock uses a 5 MHz time base, each clock tick represents 0.2 µs. For example, if you specify a count of 500, the period between updates is 100 µs (10 ksamples/s).

You can specify a count between 25 and 655,360. The corresponding period between updates ranges from 5 µs to 131 ms. The corresponding D/A clock rate ranges from 200 ksamples/s to 7.63 samples/s

Use the following formula to determine the number of counts to specify:

$$\text{counts} = \frac{\text{5 MHz time base}}{\text{update rate}}$$

For example, if you want an update rate of 1000 samples/s, specify a count of 5000, as shown in the following equation:

$$\frac{5,000,000}{1,000} = 5000$$

If the hardware cannot update the analog output channels at the exact rate determined by the count you specify, the driver calculates a rate that is as close as possible to the count you specify. To determine the actual count used by the internal D/A pacer clock, use the **K_GetClkRate** function after you start the operation. (Make sure that the frame specified in **K_GetClkRate** is a D/A frame.) Refer to page 4-62 for more information.

The internal D/A pacer clock is the default pacer clock for analog output operations. To reset the pacer clock source to the internal D/A pacer clock, use the **K_SetClk** function to set the clock source to internal. (Make sure that the frame specified in **K_SetClk** is a D/A frame.)

External Pacer Clock

You connect an external pacer clock to the XPCLK pin (pin 44) on the DAS-1701AO or DAS-1702AO board's main I/O connector. When you start an analog output operation (using **K_IntStart** or **K_DMAStart**), conversions are armed. At the next active edge of the external pacer clock (and at every subsequent active edge of the external pacer clock), the driver updates the analog output channels with values from the D/A FIFO.

To specify an external pacer clock, use the **K_SetClk** function to set the clock source to external. Then, use the **K_SetExtClkEdge** function to specify the active edge (rising or falling) of the external pacer clock. A falling edge is the default active edge for the external pacer clock.

If you are using an external pacer clock, make sure that the clock initiates updates at a rate that the DACs can handle.

Note: With the DAS-1700AO Series boards, you can use the same external pacer clock to pace both an analog input operation and an analog output operation.

Clock Used for an Analog Input Operation

If you are using either the internal A/D pacer clock or the burst mode conversion clock to pace an analog input operation on a DAS-1700AO Series board, you can use the same clock to determine the update rate for an analog output operation. This allows you to synchronize D/A conversions with A/D conversions.

The particular onboard clock used for the analog output operation depends on the conversion mode specified for the analog input operation, as follows:

- **Paced mode** - If the analog input operation is running in paced mode, the update rate for the analog output operation is determined by the internal A/D pacer clock. Refer to page 2-19 for information on setting up the internal A/D pacer clock for an analog input operation.
- **Burst mode or burst mode with SSH** - If the analog input operation is running in burst mode or burst mode with SSH, the update rate for the analog output operation is determined by the burst mode conversion clock. Refer to page 2-22 for information on setting up the burst mode conversion clock for an analog input operation.

Note: Although you can use the same external clock to pace an analog input operation and an analog output operation, you cannot synchronize an analog output operation to an analog input operation that is being paced by an external pacer clock.

To specify that you want to synchronize the analog output operation to the analog input operation, perform the following steps:

1. Use the **K_SetClk** function to specify an internal clock for the analog input operation. (Make sure that the frame specified in **K_SetClk** is an A/D frame.)
2. Use the **K_SetClk** function to specify an internal clock for the analog output operation. (Make sure that the frame specified in **K_SetClk** is a D/A frame.)
3. Use the **K_SetSync** function to specify that you want to synchronize the analog output operation with the analog input operation. (Make sure that the frame specified in **K_SetSync** is a D/A frame.)

Buffering Modes

The buffering mode determines how the driver transfers values from a user-defined buffer to the D/A FIFO. For interrupt-mode and DMA-mode analog output operations, you can specify single-cycle or continuous buffering modes. For recycle-mode analog output operations, you must specify continuous buffering mode.

The buffering modes are described in the following sections.

Single-Cycle Mode

In single-cycle mode, the driver transfers the values from the user-defined buffer to the D/A FIFO once. After all the values in the D/A FIFO have been written, the operation stops automatically. Single-cycle mode is the default buffering mode. To reset the buffering mode to single-cycle, use the **K_ClrContRun** function.

Continuous Mode

In continuous mode, the driver continuously transfers values from the user-defined buffer to the D/A FIFO until the program issues a stop function; when all the values in the buffer have been transferred, the driver transfers the values again. Use the **K_SetContRun** function to specify continuous buffering mode.

Triggers

A trigger is an event that occurs based on a specified set of conditions. For interrupt-mode, DMA-mode, and recycle-mode analog output operations, the trigger event determines when updates of the analog output channels begin.

You can specify an internal trigger, external analog trigger, or external digital trigger. The trigger event is not significant until the operation the trigger governs has been started (using **K_DMAStart** or **K_IntStart**). The point at which updates begin relative to the trigger event depends on the pacer clock; refer to page 2-50 for more information about the pacer clock.

Internal Trigger

An internal trigger is a software trigger. The trigger event occurs when you start the operation. Note that there is a slight delay between the time you start the operation and the time the trigger event occurs. The internal trigger is the default trigger source. To reset the trigger source to an internal trigger, use the **K_SetTrig** function.

External Analog Trigger

You can use the signal on any analog input channel as the trigger signal for an external analog trigger. Refer to page 2-25 for more information about an external analog trigger.

External Digital Trigger

An external digital trigger event occurs when an active edge is detected on the digital trigger signal connected to the TGIN pin (pin 46) on the board's main I/O connector. Refer to page 2-28 for more information about the external digital trigger.

If you specify an external digital trigger and you are performing a recycle-mode analog output operation (the user-defined buffer in computer memory contains no more than 2048 values and continuous buffering mode is specified), you can enable *retrigger* mode. In retrigger mode, the board starts generating the waveform stored in the D/A FIFO from the beginning each time an external digital trigger event occurs. This

mode allows you to synchronize the generation of a waveform to an external digital trigger event.

To enable retrigger mode, specify an external trigger in **K_SetTrig** and then specify the appropriate trigger polarity and sensitivity in the **K_SetDITrig** function. Refer to page 4-127 for more information.

Hardware Gates

A hardware gate is an externally applied digital signal that determines whether updates of the analog output channels occur. You connect the gate signal to the TGIN pin (pin 46) on the board's main I/O connector.

If you start an interrupt-mode, DMA-mode, or recycle-mode analog output operation (using **K_IntStart** or **K_DMAStart**) and the hardware gate is enabled, the state of the gate signal determines whether updates occur, as follows:

- If you specify a positive gate, updates occur only if the signal to TGIN is high; if the signal to TGIN is low, updates are inhibited.
- If you specify a negative gate, updates occur only if the signal to TGIN is low; if the signal to TGIN is high, updates are inhibited.

Use the **K_SetGate** function to enable and disable the hardware gate and to specify the gate polarity (positive or negative). The default state of the hardware gate is disabled.

Note: You cannot use the hardware gate with an external digital trigger. If you use a digital trigger at one point in your program and later want to use a hardware gate, you must first disable the digital trigger. You disable the digital trigger by specifying an internal trigger in **K_SetTrig** or by setting up an analog trigger (using the **K_SetADTrig** function).

Digital I/O Operations

You can perform digital I/O operations on all DAS-1700 Series boards. This section describes the following:

- Digital I/O operation modes available.
- How to access a frame.
- How to allocate and manage memory for digital I/O operations.
- How to use the digital I/O channels.
- How to specify a clock rate and a buffering mode for a digital I/O operation.

Note: You cannot use an external trigger or external pacer clock with a digital I/O operation.

Operation Modes

The operation mode determines which attributes you can specify for a digital I/O operation. You can perform digital I/O operations in single mode or interrupt mode, as described in the following sections.

Single Mode

In a single-mode digital input operation, the driver reads the value of digital input channel 0 once; in a single-mode digital output operation, the driver writes a value to digital output channel 0 once. You cannot perform any other operation until the single-mode operation is complete.

Use the **K_DIRead** function to perform a digital input operation in single mode; you specify the board you want to use, the digital input channel, and the variable in which to store the value. Use the **K_DOWrite** function to perform a digital output operation in single mode; you specify the board you want to use, the digital output channel, and the digital output value.

Note: Since digital input channel 0 is only four bits wide, you must mask the value stored by **K_DIRead** with 15 (0Fh) to obtain the actual digital input value.

The value written by **K_DOWrite** must be a 32-bit value. The four least significant bits contain the actual digital output value, and all other bits are irrelevant.

Interrupt Mode

In an interrupt-mode digital input operation, the driver reads the value of digital input channel 0 multiple times; in an interrupt-mode digital output operation, the driver writes a single value or multiple values to digital output channel 0 multiple times.

A hardware clock paces the digital I/O operation. Once the digital I/O operation begins, control returns to your program. The driver continues to read values from or write values to the digital I/O channel using an interrupt service routine.

Note: The digital input buffer and the digital output buffer each contain 16-bit integers. Each digital input value is stored in the four least significant bits of each integer in the digital input buffer.

Each digital output value is stored in the four least significant bits of each integer in the digital output buffer.

Use the **K_IntStart** function to start a digital I/O operation in interrupt mode.

You can specify either single-cycle buffering mode or continuous buffering mode for interrupt-mode operations. Refer to page 2-63 for more information on buffering modes. Use the **K_IntStop** function to stop a continuous-mode interrupt operation. Use the **K_IntStatus** function to determine the current status of an interrupt operation.

Frames

Interrupt-mode digital I/O operations require frames. Use the **K_GetDIFrame** function to access a digital input frame, called a DI frame; use the **K_GetDOFrame** function to access a digital output frame, called a DO frame. The driver returns the frame handle for the operation. Refer to page 2-6 for more information about frames.

Table 2-6 lists the elements of a DI frame; Table 2-7 lists the elements of a DO frame. The tables also list the default value of each element, the setup functions used to define each element, and the pages in this guide on which to find additional information.

Table 2-6. DI Frame Elements

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
Number of Samples	0	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
Buffering Mode	Single-cycle	K_SetContRun	page 4-125
		K_ClrContRun[2]	page 4-22
Start Channel	0	Not applicable[3]	Not applicable[3]
Stop Channel	0	Not applicable[3]	Not applicable[3]
Clock Source	Internal	Not applicable[3]	Not applicable[3]
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-123

Notes

[1] This element must be set.

[2] Use this function to reset the value of this particular frame element to its default setting without clearing the frame or getting a new frame. Whenever you clear a frame or get a new frame, this frame element is set to its default value automatically.

[3] The default value of this element cannot be changed.

Table 2-7. DO Frame Elements

Element	Default Value	Setup Function	Page Number
Buffer[1]	0 (NULL)	K_SetBuf	page 4-108
		K_SetBufI	page 4-111
Number of Samples	0	K_SetBuf	page 4-108
Start Channel	0	Not applicable[2]	Not applicable[2]
Stop Channel	0	Not applicable[2]	Not applicable[2]
Clock Source	Internal	Not applicable[2]	Not applicable[2]
Pacer Clock Rate[1]	0	K_SetClkRate	page 4-123

Notes
[1] This element must be set.
[2] The default value of this element cannot be changed.

Memory Allocation and Management

Interrupt-mode digital I/O operations use a single memory buffer to store the data to be read or written.

Since digital I/O operations typically require small arrays of data, you can reserve a memory buffer by dimensioning an array within your program's memory area. Since the DAS-1700 Series Function Call Driver reads and writes data as 16-bit integers, you must dimension all local arrays as integers.

After you dimension your array, you must assign the starting address of the array and the number of samples to store in the array.

Each supported programming language requires a particular procedure for dimensioning an array and assigning the starting address. Refer to the following pages in this guide for information:

C/C++	page 3-6
Turbo Pascal for Windows	page 3-14
Visual Basic for Windows	page 3-22

You can also use the **K_IntAlloc** function to dynamically allocate a memory buffer, if desired. You specify the operation requiring the buffer and the number of values to store in the buffer (up to a maximum of 32,767). The driver returns the starting address of the buffer and a memory handle (a unique identifier for the buffer). If you no longer need the buffer, you can free it for another use by specifying the memory handle in the **K_IntFree** function.

For Visual Basic for Windows, data in a dynamically allocated memory buffer is not directly accessible to your program. For digital input operations, you must use the **K_MoveBufToArray** function to move the data from the dynamically allocated buffer to the program's local array. (Refer to page 4-91 for more information.) For digital output operations, you must use the **K_MoveArrayToBuf** function to move the data from the program's local array to the allocated buffer. (Refer to page 4-89 for more information.)

Note: You cannot use a local array with Windows 95, 32-bit programs; you must use **K_IntAlloc** to dynamically allocate a memory buffer. You must also install the Keithley Memory Manager; refer to the *DAS-1700 Series User's Guide* for information.

Digital Input Channel

DAS-1700 Series boards contain one 4-bit digital input channel (channel 0). As shown in Figure 2-6, bit 0 contains the value of digital input line 0 (DI0); bit 1 contains the value of digital input line 1 (DI1); bit 2 contains the value of digital input line 2 (DI2); bit 3 contains the value of digital input line 3 (DI3).

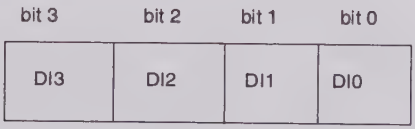


Figure 2-6. Digital Input Bits

A value of 1 in the bit position indicates that the input is high; a value of 0 in the bit position indicates that the input is low. For example, if the value is 5 (0101) on a DAS-1701ST board, the inputs at DI0 and DI2 are high and the inputs at DI1 and DI3 are low.

Note: If no signal is connected to a digital input line, the input appears high (value is 1).

Digital Output Channel

DAS-1700 Series boards contain one 4-bit digital output channel (channel 0). As shown in Figure 2-7, bit 0 contains the value to be written to digital output line 0 (DO0), bit 1 contains the value to be written to digital output line 1 (DO1), and so on.

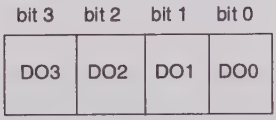


Figure 2-7. Digital Output Bits

A value of 1 in the bit position indicates that the output is high; a value of 0 in the bit position indicates that the output is low. For example, if the value written to a DAS-1701ST board is 12 (1100), the outputs at DO0 and DO1 are forced low and the outputs at DO2 and DO3 are forced high.

Pacer Clock

When performing interrupt-mode digital I/O operations, you can use the internal A/D pacer clock to determine the period between multiple reads of the digital input channel or multiple writes to the digital output channel.

Notes: You can use the internal A/D pacer clock only if it is not being used by another operation.

The actual read/write rate depends on factors such as your computer, the operating system/environment, and other software issues.

The internal A/D pacer clock uses two cascaded counters of the onboard counter/timer circuitry. The counters normally are idle. When you start the digital I/O operation (using **K_IntStart**), a value is read or written. Note that a slight time delay occurs between the time the operation is started and the time the reading or writing begins.

The counters are loaded with a count value and begin counting down. When the counters count down to 0, another value is read or written and the process repeats.

Use the **K_SetClkRate** function to specify the number of counts (clock ticks) between reads or writes.

Because the counters use a 5 MHz time base, each count represents 0.2 µs. For example, if you specify a count value of 5000, the period between reads or writes is 1 ms (1 ksamples/s); if you specify a count value of 87654, the period between reads or writes is 17.53 ms (57 samples/s).

The following table shows the valid ranges of counts (clock ticks) between updates you can specify, the corresponding periods between updates, and the corresponding internal A/D pacer clock rates. The lowest count value and smallest period between updates correspond to the highest clock rate; the highest count value and longest period between updates correspond to the lowest clock rate.

Board Series	Counts (Clock Ticks)	Periods between Updates	Internal A/D Pacer Clock Rates (samples/second)
DAS-1700ST/AO	30 to 4,294,836,225	6 µs to 14.3 minutes	166,666.667 to 0.0012
DAS-1700HR	100 to 4,294,836,225	50 µs to 14.3 minutes	50,000 to 0.0012

Note: The driver accepts a count value as low as 30 (DAS-1700ST/AO Series boards) or 100 (DAS-1700HR Series boards). However, since a FIFO is not used to buffer values for digital I/O operations, a low count value may cause overrun errors. The maximum observed read/write rates for the internal pacer clock are 1 ksamples/s when running under Windows 3.1 and Windows 3.11 for Workgroups. This rate would indicate a minimum count value of 5,000 when running under Windows 3.1 and Windows 3.11 for Workgroups.

Use the following formula to determine the number of counts to specify:

$$\text{counts} = \frac{\text{5 MHz time base}}{\text{read/write rate}}$$

For example, if you want to write data to digital output channel 0 at a rate of 500 samples/s, specify a count value of 10,000, as shown in the following equation:

$$\frac{5,000,000}{500} = 10,000$$

If the hardware cannot read or write to the digital I/O channel at the exact rate determined by the count you specify, the driver calculates a rate that is as close as possible to the count you specify. To determine the actual count used by the internal A/D pacer clock, use the **K_GetClkRate** function after you start the digital I/O operation. (Make sure that the frame specified in **K_GetClkRate** is a DI or DO frame.) Refer to page 4-62 for more information.

Buffering Modes

The buffering mode determines how the driver reads or writes the values in the buffer. For interrupt-mode digital I/O operations, you can specify single-cycle or continuous buffering modes, which are described in the following sections.

Single-Cycle Mode

In a single-cycle mode digital input operation, after the driver fills the buffer, the operation stops automatically. In a single-cycle mode digital output operation, after the driver writes the values stored in the buffer, the operation stops automatically.

Single-cycle mode is the default buffering mode. To reset the buffering mode to single-cycle mode, use the **K_ClrContRun** function.

Continuous Mode

In a continuous-mode digital input operation, the driver continuously reads digital input channel 0 and stores the values in the buffer until the program issues a stop function; any values already stored in the buffer are overwritten. In a continuous-mode digital output operation, the driver continuously writes values from the buffer to digital output channel 0 until the program issues a stop function; when all the values in the buffer have been written, the driver writes the values to digital output channel 0 again, and so forth.

Use the **K_SetContRun** function to specify continuous buffering mode.

3

Programming with the Function Call Driver

This chapter presents the following information:

- **Programming Overview** - an overview of the tasks required to write a program using the DAS-1700 Series Function Call Driver.
- **C/C++ Programming Information** - language-specific information for programming in Microsoft C/C++ (including Visual C++) and Borland C/C++.
- **Pascal Programming Information** - language-specific information for programming in Borland Turbo Pascal for Windows.
- **Visual Basic for Windows Programming Information** - language-specific information for programming in Visual Basic for Windows.

Programming Overview

To write a program using the DAS-1700 Series Function Call Driver, perform the following steps:

1. Define the application's requirements. Refer to Chapter 2 for a description of the board operations supported by the Function Call Driver and the functions that you can use to define each operation.
2. Write the program, referring to the following for information:
   - Programming flow diagrams for the preliminary tasks, on page 1-7, which illustrate the programming tasks common to all programs.
   - Programming flow diagrams for analog input operations, on page 1-8.
   - Programming flow diagrams for analog output operations, on page 1-15 (DAS-1700ST-DA Series and DAS-1702HR boards) or on page 1-18 (DAS-1700AO Series boards).
   - Programming flow diagrams for digital input operations, on page 1-26, and for digital output operations, on page 1-29.
   - Chapter 4, which contains detailed descriptions of the Function Call Driver functions.
   - The example programs in the ASO-1700 software package.
   - The FILES.TXT fle in the installation directory lists and describes the example programs.
3. Compile and link the program, referring to the following for language-specific programming information:
   - C/C++ Programming Information on page 3-3.
   - Pascal Programming Information on page 3-11.
   - Visual Basic for Windows Programming Information on page 3-17.
   - The EXAMPLES.TXT file in the installation directory for compile and link statements and other language-specific considerations for each language supported.

C/C++ Programming Information

The following sections contain information you need to allocate memory, to create a channel-gain queue, and to handle errors when programming in C or C++, as well as language-specific information for Microsoft C/C++ (including Visual C++) and Borland C/C++.

Notes: When programming in Microsoft C/C++ or Borland C/C++, proper typecasting may be required to avoid C/C++ type-mismatch warnings.

When programming in Borland C/C++, make sure that linker options are set so that case-sensitivity is disabled.

Dynamically Allocating and Assigning Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in C or C++. Refer to the example programs on disk for more information.

Notes: The code fragments for dynamically allocated memory assume that you are using DMA mode (or recycle mode); the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are using large buffers or multiple buffers, you may be limited in the amount of memory you can allocate. It is recommended that you install the Keithley Memory Manager before you begin programming to ensure that you can allocate large enough buffers. Refer to the *DAS-1700 Series User's Guide* for more information about the Keithley Memory Manager.

Allocating a Single Memory Buffer

You can use a single, dynamically allocated memory buffer for the following operations on all DAS-1700 Series boards:

- Interrupt-mode analog input operations
- Interrupt-mode analog output operations
- Interrupt-mode digital I/O operations
- DMA-mode analog input operations

For the DAS-1700AO Series boards, you can also use a single, dynamically allocated memory buffer for DMA-mode and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size `Samples` for the frame defined by `hFrame` and how to use **K_SetDMABuf** to assign the starting address of the buffer.

```
. . .
void far *AcqBuf;        //Declare pointer to buffer
WORD hMem;               //Declare word for memory handle
. . .
wDasErr = K_DMAAlloc (hFrame, Samples, &AcqBuf, &hMem);
wDasErr = K_SetDMABuf (hFrame, AcqBuf, Samples);
```

The following code illustrates how to use **K_DMAFree** later to free the allocated buffer, using the memory handle stored by **K_DMAAlloc**.

```
. . .
wDasErr = K_DMAFree (hMem);
. . .
```

Allocating Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode and DMA-mode analog input operations on all DAS-1700 Series boards. For the DAS-1700AO Series boards, you can also use multiple, dynamically allocated memory buffers for interrupt-mode, DMA-mode, and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size `Samples`, each for the frame defined by

hADFrame, and how to use **K_BufListAdd** to assign the starting addresses of the five buffers.

```
. . .
void far *AcqBuf[5];        //Declare 5 pointers to 5 buffers
WORD hMem[5];               //Declare 5 words for 5 memory handles
. . .
for (i = 0; i < 5; i++)        {
wDasErr = K_DMAAlloc (hADFrame, Samples, &AcqBuf[i], &hMem[i]);
wDasErr = K_BufListAdd (hADFrame, AcqBuf[i], Samples);
}
. . .
```

The following code illustrates how to use **K_DMAFree** later to free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
. . .
for (i = 0; i < 5; i++) {
   wDasErr = K_DMAFree (hMem[i]);
}
wDasErr = K_BufListReset (hADFrame);
```

Accessing the Data

You access the data stored in dynamically allocated buffers through C/C++ pointer indirection. For example, assume you want to display the first 10 samples of the second buffer in the multiple-buffer operation described in the previous section (AcqBuf[1]). The following code fragment illustrates how to access and display that data.

```
. . .
int huge *pData;            //Declare a pointer called pData
. . .
pData = (int huge *) AcqBuf[1]; //Assign pData to 2nd buffer
for (i = 0; i < 10; i++)
   printf ("Sample #%d %X", i, *(pData+i));
. . .
```

Note: Declaring pData as a huge pointer allows the program to directly access all data in the buffer, regardless of the buffer size.

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in C or C++. Refer to the example programs on disk for more information.

Dimensioning a Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations on all DAS-1700 Series boards. On the DAS-1700AO Series boards, you can also use a single, local array for recycle-mode analog output operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by `hFrame` and how to use **K_SetBuf** to assign the starting address of the array.

```
. . .
int Data[10000];  //Dimension array of 10,000 samples
. . .
wDasErr = K_SetBuf (hFrame, Data, 10000);
. . .
```

Dimensioning Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations on all DAS-1700 Series boards. For the DAS-1700AO Series boards, you can also use multiple, local arrays for recycle-mode analog output operations.

The following code fragment illustrates how to allocate two arrays of 32,000 samples each for the frame defined by `hADFrame`, and how to use **K_BufListAdd** to assign the starting addresses of the arrays.

```
. . .
int Data1[32000];     //Allocate Array #1 of 32,000 samples
int Data2[32000];     //Allocate Array #2 of 32,000 samples
. . .
wDasErr = K_BufListAdd (hADFrame, Data1, 32000);
wDasErr = K_BufListAdd (hADFrame, Data2, 32000);
. . .
```

Creating a Channel-Gain Queue

The DASDECL.H file defines a special data type (GainChanTable) that you can use to declare your channel-gain queue. GainChanTable is defined as follows:

```
typedef struct GainChanTable
{
   WORD num_of_codes;
   struct{
      BYTE Chan;
      char Gain;
   } GainChanAry[256];
} GainChanTable;
```

The following example illustrates how to create a channel-gain queue called `MyChanGainQueue` for a DAS-1702AO board by declaring and initializing a variable of type GainChanTable.

```
GainChanTable MyChanGainQueue =
   {8,          //Number of entries
   0, 0,        //Channel 0, gain of 1
   1, 1,        //Channel 1, gain of 2
   2, 2,        //Channel 2, gain of 4
   3, 3,        //Channel 3, gain of 8
   3, 0,        //Channel 3, gain of 1
   2, 1,        //Channel 2, gain of 2
   1, 2,        //Channel 1, gain of 4
   0, 3};       //Channel 0, gain of 8
```

After you create `MyChanGainQueue`, you must assign the starting address of `MyChanGainQueue` to the frame defined by `hFrame`, as follows:

```
wDasErr = K_SetChnGAry (hFrame, &MyChanGainQueue);
```

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Handling Errors

It is recommended that you always check the returned value (`wDasErr` in the previous examples) for possible errors. The following code fragment illustrates how to check the returned value of the **K_GetDevHandle** function.

```
. . .
if ((wDasErr = K_GetDevHandle (hDrv, BoardNum, &hDev)) ! = 0)
     {
     printf ("Error %X during K_GetDevHandle", wDasErr);
     exit (1);
     }
. . .
```

The following code fragment illustrates how to use the **K_GetErrMsg** function to access the string corresponding to an error code.

```
. . .
if ((wDasErr = K_SetChn (hAD, 2) ! = 0)
     {
     Error = K_GetErrMsg (hDev, wDasErr, &pMessage);
     printf ("%s", pMessage);
     exit (1);
     }
```

Programming in Microsoft C/C++ for Windows

The files you need to program in Microsoft C/C++ for Windows, including Microsoft Visual C++, depend on whether you are writing 16-bit or 32-bit programs. The following files are either included in the ASO-1700 software package or provided on the ASO-Win95/32-bit disk:

Windows Program	File	Description
16-bit	DASSHELL.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASDECL.H	Include file of shell function definitions (used when compiling in C)
	DASIMP.LIB	Import library of shell functions
32-bit	DASSHL32.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DASDECL.H	Include file of shell function definitions (used when compiling in C or C++)
	DASSHL32.LIB	Import library of shell functions

To create an executable file in Microsoft C/C++ for Windows, perform the following steps. (Refer to the documentation supplied with your compiler for detailed information.)

1. Create a project file.
2. Add all necessary files to the project make file. Make sure that you include *filename*.c, *filename*.rc, *filename*.def, DASIMP.LIB (or DASSHL32.LIB), where *filename* indicates the name of your program.
3. Create a stand-alone executable file (.EXE) that you can execute from within Windows.

Programming in Borland C/C++ for Windows

The files you need to program in Borland C/C++ for Windows depend on whether you are writing 16-bit or 32-bit programs. The following files are either included in the ASO-1700 software package or provided on the ASO-Win95/32-bit disk.

Windows Program	File	Description
16-bit	DASSHELL.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASDECL.H	Include file of shell function definitions (used when compiling in C)
	DASIMP.LIB	Import library of shell functions
32-bit	DASSHL32.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DASDECL.H	Include file of shell function definitions (used when compiling in C or C++)
	DASSHLBC.LIB	Import library of shell functions

To create an executable file in Borland C/C++ for Windows, perform the following steps. (Refer to the documentation supplied with your compiler for detailed information.)

1. Create a project file.
2. Add all necessary files to the project make file. Make sure that you include *filename*.c (or *filename*.cpp), *filename*.rc, *filename*.def, DASIMP.LIB (or DASSHLBC.LIB), where *filename* indicates the name of your program.
3. Make sure that you turn OFF both the **Case sensitive link** and **Case sensitive exports and imports** options.
4. Create a stand-alone executable file (.EXE) that you can execute from within Windows.

Pascal Programming Information

The following sections contain information you need to allocate memory, to create channel-gain queues, and to handle errors when programming in Pascal, as well as language-specific information for Borland Turbo Pascal for Windows.

Dynamically Allocating and Assigning Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in Pascal. Refer to the example programs on disk for more information.

Notes: The code fragments for dynamically allocated memory assume that you are using DMA mode (or recycle mode); the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are using large buffers or multiple buffers and you are programming in Borland Turbo Pascal for Windows, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate large enough buffers. Refer to the *DAS-1700 Series User's Guide* for more information about the Keithley Memory Manager.

Allocating a Single Memory Buffer

You can use a single, dynamically allocated memory buffer for interrupt-mode analog input, analog output, and digital I/O operations and for DMA-mode analog input operations on all DAS-1700 Series boards. For the DAS-1700AO Series boards, you can also use a single, dynamically allocated memory buffer for DMA-mode and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size `Samples` for the frame defined by `hFrame` and how to use **K_SetDMABuf** to assign the starting address of the buffer.

It is recommended that you declare a dummy type array of ^Integer. The dimension of this array is irrelevant; it is used only to satisfy Pascal's type-checking requirements.

```
Type
   IntArray = Array[0..1] of Integer;
. . .
Var
   AcqBuf : ^IntArray;    { Declare buffer of dummy type }
   hMem : Word;    { Declare word for memory handle, hMem }
. . .
wDasErr := K_DMAAlloc (hFrame, Samples, @AcqBuf, hMem);
wDasErr := K_SetDMABuf (hFrame, AcqBuf, Samples);
```

The following code illustrates how to use **K_DMAFree** to free the allocated buffer later, using the memory handle stored by **K_DMAAlloc**.

```
. . .
wDasErr := K_DMAFree (hMem);
. . .
```

Allocating Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode and DMA-mode analog input operations on all DAS-1700 Series boards. On the DAS-1700AO Series boards, you can also use multiple, dynamically allocated memory buffers for interrupt-mode, DMA-mode, and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size `Samples` each for the frame defined by `hADFrame`, and how to use **K_BufListAdd** to assign the starting addresses of the five buffers.

It is recommended that you declare a dummy type array of ^Integer. The dimension of this array is irrelevant; it is used only to satisfy Pascal's type-checking requirements.

```
. . .
Type
IntArray = Array[0..1] of Integer;
. . .
Var
AcqBuf : Array[0..4] of ^IntArray;{5 buffers, dummy type}
hMem : Array[0..4] of Word; {5 words for 5 memory handles}
. . .
For i := 0 to 4 do begin
   wDasErr := K_DMAAlloc (hADFrame, Samples, @AcqBuf[i], hMem[i]);
   wDasErr := K_BufListAdd (hADFrame, AcqBuf[i], Samples);
End;
```

The following code illustrates how to use **K_DMAFree** later to free the allocated buffers, using the memory handles stored by **K_DMAAlloc**; if you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
. . .
For i := 0 to 4 do begin
   wDasErr := K_DMAFree (hMem[i]);
End;
wDasErr := K_BufListReset (hADFrame);
. . .
```

Accessing the Data

You access the data stored in dynamically allocated buffers through Pascal pointer indirection. For example, assume that you want to display the first 10 samples of the second buffer in the multiple-buffer operation described in the previous section (`AcqBuf[1]`). The following code fragment illustrates how to access and display the data.

```
. . .
for i := 0 to 10 do begin
   writeln ('Sample #', i,' =', AcqBuf[1]^[i]);
End;
. . .
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in Pascal. Refer to the example programs on disk for more information.

Dimensioning a Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations on all DAS-1700 Series boards. On DAS-1700AO Series boards, you can also use a single, local array for recycle-mode analog output operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by `hFrame`, and how to use **K_SetBuf** to assign the starting address of the array.

```
. . .
Data : Array[0..9999] of Integer;
. . .
wDasErr := K_SetBuf (hFrame, @Data[0], 10000);
. . .
```

Dimensioning Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations on all DAS-1700 Series boards. On the DAS-1700AO Series

boards, you can also use multiple, local arrays for interrupt-mode and recycle-mode analog output operations.

The following code fragment illustrates how to allocate two arrays of 32,000 samples each for the frame defined by `hADFrame`, and how to use **K_BufListAdd** to assign the starting addresses of the arrays.

```
. . .
Data1 : Array[0..31999] of Integer; { Allocate Array #1 }
Data2 : Array[0..31999] of Integer; { Allocate Array #2 }
. . .
wDasErr := K_BufListAdd (hADFrame, @Data1[0], 32000);
wDasErr := K_BufListAdd (hADFrame, @Data2[0], 32000);
. . .
```

Creating a Channel-Gain Queue

The following example illustrates how to create a channel-gain queue called `MyChanGainQueue` for a DAS-1702ST board by defining a Record as a new type. You must use **K_SetChnGAry** to assign the starting address of `MyChanGainQueue` to the frame defined by `hFrame`.

```
Type
   GainChanTable = Record
   num_of_codes : Integer;
   queue : Array[0..255] of Byte;
   end;
. . .
Const
   MyChanGainQueue : GainChanTable = (
   num_of_codes : (8);      { Number of entries }
   queue :(0, 0,   { Channel 0, gain of 1 }
           1, 1,       { Channel 1, gain of 2 }
           2, 2,       { Channel 2, gain of 4 }
           3, 3,       { Channel 3, gain of 8 }
           3, 0,       { Channel 3, gain of 1 }
           2, 1,       { Channel 2, gain of 2 }
           1, 2,       { Channel 1, gain of 4 }
           0, 3)       { Channel 0, gain of 8 }
         );
         . . .
wDasErr := K_SetChnGAry (hFrame, MyChanGainQueue.num_of_codes);
```

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Handling Errors

It is recommended that you always check the returned value (`wDasErr` in the previous examples) for possible errors. The following code fragment illustrates how to check the returned value of the **K_GetDevHandle** function.

```
...
wDasErr := K_GetDevHandle (hDrv, 0, hDev);
if wDasErr <> 0 then
BEGIN
   FormatStr (HexErr, ' %4x ', wDasErr);
   writeln ('Error', HexErr, 'during K_GetDevHandle');
   Halt (1);
END;
...
```

Programming in Borland Turbo Pascal for Windows

To program in Borland Turbo Pascal for Windows (16-bit programming only), you need the following files. These files are provided in the ASO-1700 software package.

File	Description
DASSHELL.DLL	Dynamic Link Library of shell functions
DASSUPRT.DLL	Dynamic Link Library of support functions
DASDECL.INC	Include file

To create an executable file in Borland Turbo Pascal for Windows, perform the following steps:

1. Load *filename*.pas into the Borland Turbo Pascal for Windows environment, where *filename* indicates the name of your program.
2. Create an executable file (.EXE).

Visual Basic for Windows Programming Information

The following sections contain information you need to allocate memory, to create channel-gain queues, to convert integer data for digital I/O operations, and to handle errors when programming in Visual Basic for Windows.

Dynamically Allocating and Assigning Memory Buffers

This section provides code fragments that describe how to allocate and assign dynamically allocated memory buffers when programming in Visual Basic for Windows. Refer to the example programs on disk for more information.

Notes: The code fragments for dynamically allocated memory assume that you are using DMA mode (or recycle mode); the code for interrupt mode is identical, except that you use the appropriate interrupt-mode functions instead of the DMA-mode functions.

If you are using large buffers or multiple buffers, you may be limited in the amount of memory you can allocate. It is recommended that you use the Keithley Memory Manager before you begin programming to ensure that you can allocate large enough buffers. Refer to the *DAS-1700 Series User's Guide* for more information about the Keithley Memory Manager.

Allocating a Single Memory Buffer

You can use a single, dynamically allocated memory buffer for the following operations on all DAS-1700 Series boards:

- Interrupt-mode analog input, analog output, and digital I/O operations
- DMA-mode analog input operations

On the DAS-1700AO Series boards, you can also use a single, dynamically allocated memory buffer for DMA-mode and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate a buffer of size `Samples` for the frame defined by `hFrame` and how to use **K_SetDMABuf** to assign the starting address of the buffer.

```
. . .
Global AcqBuf As Long    ' Declare pointer to buffer
Global hMem As Integer   ' Declare integer for memory handle
. . .
wDasErr = K_DMAAlloc (hFrame, Samples, AcqBuf, hMem)
wDasErr = K_SetDMABuf (hFrame, AcqBuf, Samples)
. . .
```

The following code illustrates how to use **K_DMAFree** to free the allocated buffer later, using the memory handle stored by **K_DMAAlloc**.

```
. . .
wDasErr = K_DMAFree (hMem)
. . .
```

Allocating Multiple Memory Buffers

You can use multiple, dynamically allocated memory buffers for interrupt-mode and DMA-mode analog input operations on all DAS-1700 Series boards. On the DAS-1700AO Series boards, you can also use multiple, dynamically allocated memory buffers for interrupt-mode, DMA-mode, and recycle-mode analog output operations.

The following code fragment illustrates how to use **K_DMAAlloc** to allocate five buffers of size `Samples` each for the frame defined by `hADFrame`, and how to use **K_BufListAdd** to assign the starting addresses of the five buffers.

```
. . .
Global AcqBuf(4) As Long          ' Declare 5 pointers to 5 buffers
Global hMem(4) As Integer         ' Declare 5 memory handles
. . .
for i% = 0 to 4
   wDasErr = K_DMAAlloc (hFrame, Samples, AcqBuf(i%), hMem(i%))
   wDasErr = K_BufListAdd (hFrame, AcqBuf(i%), Samples)
next i%
. . .
```

The following code illustrates how to use **K_DMAFree** to free the allocated buffers later, using the memory handles stored by **K_DMAAlloc**. If you free the allocated buffers, you must also use **K_BufListReset** to reset the buffer list associated with the frame.

```
. . .
for i% = 0 to 4
   wDasErr = K_DMAFree (hMem(i%))
next i%
wDasErr = K_BufListReset (hADFrame)
. . .
```

Accessing Data from a Dynamically Allocated Memory Buffer with Fewer than 64 KB of Data

In Visual Basic for Windows, you cannot directly access analog input samples stored in dynamically allocated memory buffers. You must use **K_MoveBufToArray** to move a subset (up to 32,767 samples) of the data into a local array as required.

The following code fragment illustrates how to move the first 100 samples of the second buffer in the multiple-buffer operation described in the previous section (`AcqBuf(1)`) to a local array.

```
. . .
Dim Buffer(1000) As Integer   ' Declare local memory buffer
. . .
wDasErr = K_MoveBufToArray (Buffer(0), AcqBuf(1), 100)
. . .
```

Accessing Data from a Dynamically Allocated Memory Buffer with More than 64 KB of Data

When Windows is running, the CPU operates in protected mode. Selectors are used to index into tables to translate virtual memory addresses to physical memory addresses. No mathematical relationship exists between a selector and the memory location with which it is associated.

When a memory buffer of more than 64 KB (32,767 values) is used, multiple selectors are required. Assuming that you are using the Keithley Memory Manager, **K_IntAlloc** uses a "tiled" method for allocating memory whereby a mathematical relationship does exist among the

selectors. Specifically, if you allocate a buffer of more than 64 KB, each selector that is allocated has an arithmetic value that is eight greater than the previous one. The format of the address is a 32-bit value whose high word is the 16-bit selector value and low word is the 16-bit offset value. When the offset reaches 64 KB, the next consecutive memory address location can be accessed by adding eight to the selector and resetting the offset to 0. To do this, add &h80000 to the buffer starting address.

Table 3-1 illustrates the mapping of consecutive memory locations in protected-mode "tiled" memory, where *xxxxxxxx* indicates the address calculated by the CPU memory mapping mechanism.

Table 3-1. Protected-Mode Memory Architecture

Selector:Offset	32-Bit Linear Address
. . . . :	. . . . .
32E6:FFFE	*xxxxxxxx*
32E6:FFFF	*xxxxxxxx* + 1
32EE:0000	*xxxxxxxx* + 2
32EE:0001	*xxxxxxxx* + 3
. . . . :	. . . . .

The following code fragment illustrates moving 1,000 values from a memory buffer (`AcqBuf`) allocated with 50,000 values to the program's local array (`Array`), starting at the sample at buffer index 40,000. First, start with the buffer address passed in **K_SetBuf**. Then, determine how deep into the buffer (in 64 KB pages) the desired starting sample is located and add &h80000 to the buffer address for each 64 KB page. Finally, add any additional offset after the 64 KB pages to the buffer address.

```
Dim AcqBuf As Long
Dim NumSamps As Long

Dim Array(1000) As Integer

NumSamps = 50000
wDasErr = K_IntAlloc (hFrame, NumSamps, AcqBuf, hMem)
. . .
```

```
'Acquisition routine
. . .
DesiredSamp = 40000
DesiredByte = DesiredSamp * 2        'Number of bytes into buffer
AddSelector = DesiredByte / &h10000 'Number of 64K pages into buffer
RemainingOffset = DesiredByte Mod &h10000    'Additional offset

DesiredBuffLoc = AcqBuf + (AddSelector * &h80000) + RemainingOffset

wDasErr = K_MoveBufToArray (Array(0), DesiredBuffLoc, 1000)
```

Accessing More than 64 KB of Data from a Dynamically Allocated Buffer

To move more than 64 KB of data (32,767 values) from the memory buffer to the program's local array, the program must call **K_MoveBufToArray** more than once. For example, assume that `pBuf` is a pointer to a dynamically allocated buffer that contains 65,536 values. The following code fragment illustrates how to move 65,536 values from the dynamically allocated buffer to the local array within the program:

```
...
Dim Data [3, 16384] As Integer
...
wDasErr = K_MoveBufToArray (Data(0,0), pBuf, 16384)

'Same selector, add 32,768 bytes to offset: add &h8000
wDasErr = K_MoveBufToArray (Data(1,0), pBuf + &h8000, 16384)

'Add 8 to selector, offset = 0: add &h80000
wDasErr = K_MoveBufToArray (Data(2,0), pBuf + &h80000, 16384)

'Add 8 to selector, add 32,768 bytes to offset: add &h88000
wDasErr = K_MoveBufToArray (Data(3,0), pBuf + &h88000, 16384)
```

Dimensioning and Assigning Local Arrays

This section provides code fragments that describe how to dimension and assign local arrays when programming in Visual Basic for Windows. Refer to the example programs on disk for more information.

Dimensioning a Single Array

You can use a single, local array for interrupt-mode analog input, analog output, and digital I/O operations on all DAS-1700 Series boards. On DAS-1700AO Series boards, you can also use a single, local array for recycle-mode analog output operations.

The following code fragment illustrates how to dimension an array of 10,000 samples for the frame defined by `hFrame` and how to use **K_SetBufI** to assign the starting address of the array.

```
. . .
Global Data(9999) As Integer      ' Allocate array
. . .
wDasErr = K_SetBufI (hFrame, Data(0), 10000)
. . .
```

Dimensioning Multiple Arrays

You can use multiple, local arrays for interrupt-mode analog input operations on all DAS-1700 Series boards. On the DAS-1700AO Series boards, you can also use multiple, local arrays for interrupt-mode analog output operations and for recycle-mode analog output operations that you perform using **K_IntStart**.

The following code fragment illustrates how to dimension two arrays of 32,000 samples each for the frame defined by `hADFrame` and how to use **K_BufListAdd** to assign the starting addresses of the arrays.

```
. . .
Global Data1(31999) As Integer    ' Allocate Array #1
Global Data2(31999) As Integer    ' Allocate Array #2
. . .
wDasErr = K_BufListAdd (hADFrame, Data1(0), 32000)
wDasErr = K_BufListAdd (hADFrame, Data2(0), 32000)
. . .
```

Note: The declaration of the second parameter of the **K_BufListAdd** function in the DASDECL.BAS file is not applicable to local arrays. You must change the declaration of the *pBuf* from ByVal *pBuf* As Long to *pBuf* As Integer.

Creating a Channel-Gain Queue

Before you create your channel-gain queue, you must declare an array of integers to accommodate the required number of entries. It is recommended that you declare arrays that are two times the number of entries plus one For example, to accommodate the maximum possible channel-gain queue (256 entries), declare an array of 513 integers ((256 x 2) + 1).

Next, you must fill the array with the channel-gain information. After you create the channel-gain queue, use **K_FormatChnGAry** to reformat the channel-gain queue so that it can be used by the DAS-1700 Series Function Call Driver.

The following code fragment illustrates how to create a four-entry channel-gain queue called `MyChanGainQueue` for a DAS-1702ST board and how to use **K_SetChnGAry** to assign the starting address of `MyChanGainQueue` to the frame defined by `hFrame`.

```
. . .
Global MyChanGainQueue(513) As Integer 'Maximum # of entries
. . .
MyChanGainQueue(0) = 4      ' Number of channel-gain pairs
MyChanGainQueue(1) = 0      ' Channel 0
MyChanGainQueue(2) = 0      ' Gain of 1
MyChanGainQueue(3) = 1      ' Channel 1
MyChanGainQueue(4) = 1      ' Gain of 2
MyChanGainQueue(5) = 2      ' Channel 2
MyChanGainQueue(6) = 2      ' Gain of 4
MyChanGainQueue(7) = 2      ' Channel 2
MyChanGainQueue(8) = 3      ' Gain of 8
. . .
wDasErr = K_FormatChnGAry (MyChanGainQueue(0))
wDasErr = K_SetChnGAry (hFrame, MyChanGainQueue(0))
. . .
```

Your Visual Basic for Windows program can no longer read the channel-gain queue once it is formatted. To read or modify the array after it has been formatted, you must use **K_RestoreChnGAry** as follows:

```
. . .
wDasErr = K_RestoreChnGAry (MyChanGainQueue(0))
. . .
```

When you start the next analog input operation (using **K_IntStart** or **K_DMAStart**), channel 0 is sampled at a gain of 1, channel 1 is sampled at a gain of 2, channel 2 is sampled at a gain of 4, and so on.

Converting 16-Bit Values (DAS-1700HR Series)

DAS-1700HR Series boards use a 16-bit, twos complement data format. Visual Basic for Windows treats integers as signed numbers. When reading an analog input value from a DAS-1700HR Series board, a value greater than 32,767 is represented as a negative integer. When writing a trigger level or hysteresis value, you must represent a value greater than 32,767 as a negative integer. The following code fragment illustrates how to convert analog input values from a DAS-1700HR Series board.

```
Dim lVal As Long
Dim pVal As Integer
Dim volts As Single
Dim span As Single
Dim ADMode As Integer
. . .
wDasErr = K_GetADMode (hDev, ADMode)
wDasErr = K_ADRead (hDev, 0, 0, pVal)
IF (ADMode = 0) THEN    'Bipolar mode
   volts = pVal * (span / 65536)
ELSE           'In unipolar mode, adjust the data
   IF (pVal < 0) THEN
      lVal = pVal + 65536  'Convert negative number to
                           positive
   ELSE
      lVal = pVal
   ENDIF
   volts = lVal * (span / 65536)
ENDIF
```

Handling Errors

It is recommended that you always check the returned value (`wDasErr` in the previous examples) for possible errors. The following code fragment illustrates how to check the returned value of the **K_GetDevHandle** function.

```
. . .
wDasErr = K_GetDevHandle (hDrv, BoardNum, hDev)
If (wDasErr <> 0) Then
   MsgBox "K_GetDevHandle Error: "+ Hex$ (wDasErr),
         MB_ICONSTOP, "DAS-1700 ERROR"
   End
End If
. . .
```

Programming in Visual Basic for Windows

The files you need to program in Visual Basic for Windows, depend on whether you are writing a 16-bit or 32-bit program. The following files are either included in the ASO-1700 software package or provided on the ASO-Win95/32-bit disk.

Windows Program	File	Description
16-bit	DASSHELL.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASDECL.BAS	Include file of shell function definitions
32-bit	DASSHL32.DLL	Dynamic Link Library of shell functions
	DASSUPRT.DLL	Dynamic Link Library of support functions
	DASSHL16.DLL	Dynamic Link Library of support functions
	DASDEC32.BAS	Include file of shell function definitions

To create an executable file from the Visual Basic for Windows environment, perform the following steps. Refer to the documentation supplied with your compiler for complete information.

1. Start Visual Basic for Windows.
2. Add the necessary include files to the project: DASDECL.BAS (or DASDEC32.BAS).
3. Create an executable file (.EXE).

4

Function Reference

The FCD functions are organized into the following groups:

- Initialization functions
- Operation functions
- Frame management functions
- Memory management functions
- Buffer address functions
- Buffering mode functions
- Conversion mode functions
- Channel and gain functions
- Clock functions
- Trigger functions
- Gate functions
- Miscellaneous functions

The particular functions associated with each function group are presented in Table 4-1. The remainder of this chapter presents detailed descriptions of all the FCD functions, arranged in alphabetical order.

Table 4-1. Functions

Function Type	Function Name	Page Number
Initialization	K_OpenDriver	page 4-93
	K_CloseDriver	page 4-16
	K_GetDevHandle	page 4-66
	K_FreeDevHandle	page 4-49
	K_DASDevInit	page 4-24
Operation	K_ADRead	page 4-7
	K_DAWriteGain	page 4-26
	K_DIRead	page 4-29
	K_DOWrite	page 4-46
	K_DMAStart	page 4-36
	K_DMAStatus	page 4-38
	K_DMAStop	page 4-43
	K_IntStart	page 4-81
	K_IntStatus	page 4-83
	K_IntStop	page 4-87
Frame management	K_GetADFrame	page 4-56
	K_GetDAFrame	page 4-64
	K_GetDIFrame	page 4-68
	K_GetDOFrame	page 4-70
	K_FreeFrame	page 4-51
	K_ClearFrame	page 4-14

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Memory management	K_DMAAlloc	page 4-31
	K_DMAFree	page 4-34
	K_IntAlloc	page 4-77
	K_IntFree	page 4-80
	K_MoveArrayToBuf	page 4-89
	K_MoveBufToArray	page 4-91
Buffer address	K_SetBuf	page 4-108
	K_SetBufI	page 4-111
	K_SetDMABuf	page 4-129
	K_BufListAdd	page 4-9
	K_BufListReset	page 4-12
Buffering mode	K_ClrContRun	page 4-22
	K_SetContRun	page 4-125
Conversion mode	K_SetADFreeRun	page 4-102
	K_ClrADFreeRun	page 4-20
	K_SetSSH	page 4-137

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Channel and gain	K_SetChn	page 4-115
	K_SetStartStopChn	page 4-139
	K_SetG	page 4-133
	K_SetStartStopG	page 4-141
	K_SetChnGAry	page 4-117
	K_FormatChnGAry	page 4-48
	K_RestoreChnGAry	page 4-95
	K_SetADCommonMode	page 4-98
	K_SetADConfig	page 4-100
	K_SetADMode	page 4-104
	K_GetADCommonMode	page 4-52
	K_GetADConfig	page 4-54
	K_GetADMode	page 4-58
Clock	K_SetClk	page 4-120
	K_SetClkRate	page 4-123
	K_GetClkRate	page 4-62
	K_SetBurstTicks	page 4-113
	K_GetBurstTicks	page 4-60
	K_SetExtClkEdge	page 4-131
	K_SetSync	page 4-144
Trigger	K_SetTrig	page 4-146
	K_SetADTrig	page 4-106
	K_SetTrigHyst	page 4-148
	K_SetDITrig	page 4-127
	K_SetAboutTrig	page 4-96
	K_ClrAboutTrig	page 4-18

Table 4-1. Functions (cont.)

Function Type	Function Name	Page Number
Gate	K_SetGate	page 4-135
Miscellaneous	K_GetErrMsg	page 4-72
	K_GetVer	page 4-75
	K_GetShellVer	page 4-73

Keep the following conventions in mind when reading this chapter:

- Under "Boards Supported," *All* refers to the following boards: DAS-1701ST, DAS-1701ST-DA, DAS-1702ST, DAS-1702ST-DA, DAS-1702HR, DAS-1702HR-DA, DAS-1701AO, and DAS-1702AO.
- The data types DWORD, WORD, and BYTE are defined in the language-specific include files.
- Variable names are shown in italics.
- The return value for all DAS-1700 Series FCD functions is the error/status code. Error/status code zero indicates that the function executed successfully. A nonzero error/status code indicates that an error occurred. Refer to Appendix A for more information.
- In the usage section, the variables are not defined. It is assumed that they are defined as shown in the prototype. The name of each variable in both the prototype and the usage sections includes a prefix that indicates the associated data type. These prefixes are described in Table 4-2.

Table 4-2. Data Type Prefixes

Prefix	Data Type	Comments
sz	Pointer to string terminated by zero	This data type typically is used for variables that specify the driver's configuration file name.
h	Handle to device, frame, and memory block	This data type is used for handle-type variables. You declare handle-type variables in your program as long or DWORD, depending on the language you are using. The actual variable is passed to the driver by value.
ph	Pointer to a handle-type variable	This data type is used when calling the FCD functions to get a driver handle, a device handle, a frame handle, or a memory handle. The actual variable is passed to the driver by reference.
p	Pointer to a variable	This data type is used for pointers to all types of variables, except handles (h). It is typically used when passing a parameter of any type to the driver by reference.
n	Number value	This data type is used when passing a number, typically a byte, to the driver by value.
w	16-bit word	This data type typically is used when passing an unsigned integer to the driver by value.
a	Array	This data type typically is used in conjunction with other prefixes listed here; for example, *anVar* denotes an array of numbers.
f	Float	This data type denotes a single-precision floating-point number.
d	Double	This data type denotes a double-precision floating-point number.
dw	32-bit double word	This data type typically is used when passing an unsigned long to the driver by value.

Boards Supported

All

Purpose

Reads a single analog input value.

Prototype

C/C++

DASErr far pascal K_ADRead (DWORD *hDev*, BYTE *nChan*, BYTE *nGain*, void far **pData*);

Turbo Pascal for Windows

Function K_ADRead (*hDev* : Longint; *nChan* : Byte; *nGain* : Byte; *pData* : Pointer) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_ADRead Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nChan* As Integer, ByVal *nGain* As Integer, *pData* As Integer) As Integer

Parameters

hDev — Handle associated with the board.

nChan — Analog input channel.
Valid values:

No EXP-1800 expansion accessories attached	Differential: **0** to **7** Single-ended: **0** to **15**
n EXP-1800 expansion accessories attached	**0** to **15**(***n*** + **1**)

nGain — Gain code.
Valid values: **0** to **3** for DAS board channels
0 to **7** for EXP-1800 channels

Refer to Table 2-2 on page 2-13 for the gains and input ranges associated with each gain code.

pData — Acquired analog input value.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function reads the analog input channel *nChan* on the board specified by *hDev* at the gain represented by *nGain*, and stores the count in *pData*.

The data stored in *pData* is a count value. Refer to Appendix A for information on converting the count value to a voltage.

For DAS-1700HR Series boards, the count value stored in *pData* can range from 0 to 65,535. Since Visual Basic for Windows treats integers as signed numbers, values greater than 32,767 are represented as negative integers. For information on converting these values for the DAS-1700HR Series boards, refer to page 3-24.

See Also

K_DMAStart, K_IntStart

Usage

C/C++

```
#include "DASDECL.H"
...
int wADValue;
...
wDasErr = K_ADRead (hDev, 0, 0, &wADValue);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wADValue : Integer;
...
wDasErr := K_ADRead (hDev, 0, 0, @wADValue);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wADValue As Integer
...
wDasErr = K_ADRead (hDev, 0, 0, wADValue)
```

Boards Supported All

Purpose Adds a buffer or an array to the list of multiple buffers/arrays.

Prototype

C/C++
DASErr far pascal K_BufListAdd (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal for Windows
Function K_BufListAdd (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_BufListAdd Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *pBuf* As Long, ByVal *dwSamples* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

pBuf Starting address of buffer.

dwSamples Number of samples in the buffer.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	Interrupt	**1** to **5000000**
		DMA	**1** to **65536**
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	**1** to **5000000**
	DAS-1700AO Series	Interrupt	**1** to **5000000**
		DMA	**1** to **65536**
		Recycle	**1** to **2048**
Digital I/O	All	Interrupt	**1** to **5000000**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function adds the buffer or array at the address pointed to by *pBuf* to the list of multiple buffers/arrays; the number of samples in the buffer/array is specified in *dwSamples*.

The driver supports multiple buffers/arrays for analog input operations on all DAS-1700 Series boards. For the DAS-1700AO Series boards, the driver also supports multiple buffers/arrays for analog output operations.

Before you add the buffer or array to the list, you must either allocate the buffer dynamically (using **K_IntAlloc** or **K_DMAAlloc)**, or dimension the array locally.

Make sure that you add buffers or arrays to the list in the order in which you want to use them: Buffer 1 first, Buffer 2 second, and so on. You can add up to 149 buffers. Use **K_IntStatus** or **K_DMAStatus** to determine which buffer is currently in use.

If you are programming in Visual Basic for Windows and you are using this function to specify the starting address of a local array, the declaration of *pBuf* in the DASDECL.BAS file is not applicable. In this case, you must change the declaration of *pBuf* from ByVal *pBuf* As Long to Seg *pBuf* As Integer.

See Also

K_BufListReset, K_DMAAlloc, K_IntAlloc

Usage

C/C++

```
#include "DASDECL.H"
...
void far *pBuf[5];   // Buffer pointers
WORD hMem[5];   // Buffer handles
...
for (i = 0; i < 5; i++) {
   wDasErr = K_DMAAlloc (hAD, 1000, &pBuf[i], &hMem[i]);
   wDasErr = K_BufListAdd (hAD, pBuf[i], 1000);
}
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;   { Buffer pointers }
hMem : Array [0..4] of Word;   { Buffer handles }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, 1000, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], 1000);
END;
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf(5) As Long    ' Buffer pointers
Global hMem(5) As Integer    ' Buffer handles
...
For I% = 0 To 4
   wDasErr = K_DMAAlloc (hAD, 1000, pBuf(I%), hMem(I%))
   wDasErr = K_BufListAdd (hAD, pBuf(I%), 1000)
Next I%
```

K_BufListReset

Boards Supported

All

Purpose

Clears the list of multiple buffers/arrays.

Prototype

C/C++

DASErr far pascal K_BufListReset (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_BufListReset (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_BufListReset Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function clears all buffers or arrays from the list of multiple buffers/arrays.

This function does not deallocate the buffers in the list. If you no longer need dynamically allocated buffers, use **K_IntFree** or **K_DMAFree** to free the buffers before resetting the buffer list.

See Also

K_BufListAdd, K_DMAFree, K_IntFree

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_BufListReset (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_BufListReset (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_BufListReset (hAD)
```

K_ClearFrame

Boards Supported

All

Purpose

Sets the elements of a frame to their default values.

Prototype

C/C++
DASErr far pascal K_ClearFrame (DWORD *hFrame*);

Turbo Pascal for Windows
Function K_ClearFrame (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClearFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function sets the elements of the frame specified by *hFrame* to their default values.

See Also

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame

Refer to the following tables for the default values of the elements of each type of frame:

Frame Type	Applicable Boards	Refer to
A/D	All	Table 2-1 on page 2-8
D/A	DAS-1700ST-DA Series DAS-1702HR-DA	Table 2-3 on page 2-34
	DAS-1700AO Series	Table 2-4 on page 2-43
DI	All	Table 2-6 on page 2-58
DO	All	Table 2-7 on page 2-59

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_ClearFrame (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_ClearFrame (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClearFrame (hAD)
```

K_CloseDriver

Boards Supported

All

Purpose

Closes a previously initialized Keithley DAS Function Call Driver.

Prototype

C/C++

DASErr far pascal K_CloseDriver (DWORD *hDrv*);

Turbo Pascal for Windows

Function K_CloseDriver (*hDrv* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_CloseDriver Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long) As Integer

Parameters

hDrv	Driver handle you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the driver handle specified by *hDrv* and closes the associated use of the Function Call Driver. This function also frees all device handles and frame handles associated with *hDrv*.

If *hDrv* is the last driver handle specified for the Function Call Driver, the driver is shut down and unloaded.

See Also

K_OpenDriver

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_CloseDriver (hDrv);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_CloseDriver (hDrv);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_CloseDriver (hDrv)
```

K_ClrAboutTrig

Boards Supported

All

Purpose

Disables the about trigger for an analog input operation.

Prototype

C/C++

DASErr far pascal K_ClrAboutTrig (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_ClrAboutTrig (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_ClrAboutTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function disables the about trigger for the operation defined by *hFrame*.

K_GetADFrame and **K_ClearFrame** also disable the about trigger.

See Also

K_SetAboutTrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_ClrAboutTrig (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_ClrAboutTrig (hAD);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClrAboutTrig (hAD)
```

K_ClrADFreeRun

Boards Supported

All

Purpose

Specifies paced conversion mode for an analog input operation.

Prototype

C/C++

DASErr far pascal K_ClrADFreeRun (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_ClrADFreeRun (*hFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_ClrADFreeRun Lib "DASSHELL.DLL" (ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function sets the conversion mode for the operation defined by *hFrame* to paced mode and sets the Conversion Mode element in the frame accordingly.

K_GetADFrame and **K_ClearFrame** also enable paced conversion mode.

See Also

K_SetADFreeRun

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_ClrADFreeRun (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_ClrADFreeRun (hAD);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClrADFreeRun (hAD)
```

K_ClrContRun

Boards Supported

All

Purpose

Specifies single-cycle buffering mode.

Prototype

C/C++
DASErr far pascal K_ClrContRun (DWORD *hFrame*);

Turbo Pascal for Windows
Function K_ClrContRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_ClrContRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function sets the buffering mode for the operation defined by *hFrame* to single-cycle mode and sets the Buffering Mode element in the frame accordingly.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame,** and **K_ClearFrame** also enable single-cycle buffering mode.

See Also

K_SetContRun

For more information on buffering modes, refer to the following pages:

Operation	Applicable Boards	Refer to
Analog input	All DAS-1700 Series	page 2-23
Analog output	DAS-1700ST-DA Series, DAS-1702HR-DA	page 2-39
	DAS-1700AO	page 2-53
Digital I/O	All DAS-1700 Series	page 2-63

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_ClrContRun (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_ClrContRun (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_ClrContRun (hAD)
```

K_DASDevInit

Boards Supported

All

Purpose

Reinitializes a board.

Prototype

C/C++

DASErr far pascal K_DASDevInit (DWORD *hDev*);

Turbo Pascal for Windows

Function K_DASDevInit (*hDev* : Longint) : Longint; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DASDevInit Lib "DASSHELL.DLL"
(ByVal *hDev* As Long) As Integer

Parameters

hDev	Handle associated with the board.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function reinitializes the board specified by *hDev* after it was initialized with **K_GetDevHandle**.

Before you can use this function, you must stop all operations on the board.

See Also

K_GetDevHandle

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_DASDevInit (hDev);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_DASDevInit (hDev);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_DASDevInit (hDev)
```

K_DAWriteGain

Boards Supported

DAS-1701ST-DA, DAS-1702ST-DA, DAS-1702HR-DA, DAS-1701AO, DAS-1702AO,

Purpose

Writes a single analog output value.

Prototype

C/C++

DASErr far pascal K_DAWriteGain (DWORD *hDev*, WORD *nChan*, WORD *nGain*, DWORD *dwData*);

Turbo Pascal for Windows

Function K_DAWriteGain (*hDev* : Longint; *nChan* : Word; *nGain* : Word; *dwData* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DAWriteGain Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nChan* As Integer, ByVal *nGain* As Integer, ByVal *dwData* As Long) As Integer

Parameters

hDev — Handle associated with the board.

nChan — Analog output channel.
Valid values:

DAS-1700ST-DA Series	**0** = Channel 0 **1** = Channel 1 **2** = Channel 2 **3** = Channel 3
DAS-1702HR-DA DAS-1700AO Series	**0** = Channel 0 **1** = Channel 1

nGain — Gain code (for analog output range).
Valid values:

DAS-1700ST-DA Series DAS-1702HR-DA	**0** (±10 V)
DAS-1700AO Series	**0** (±5 V) **1** (±10 V)

dwData Analog output value.
Valid values:

DAS-1700ST-DA Series DAS-1700AO Series	**–2048** to **2047**
DAS-1702HR-DA	**–32768** to **32767**

Return Value Error/status code. Refer to Appendix A.

Remarks This function writes the value *dwData* to the analog output channel specified by *nChan* with the range specified by *nGain* on the board specified by *hDev*.

dwData is a 32-bit variable. For DAS-1700ST-DA Series and the DAS-1700AO Series boards, the output value uses only the least significant 12 bits; for the DAS-1702HR-DA board, the output value uses only the least significant 16 bits.

The data stored in *dwData* is a count value. Refer to Appendix B for information on converting a voltage value to a count.

For the DAS-1700ST-DA Series and DAS-1702HR-DA boards, the driver actually ignores the *nGain* parameter and uses the only range available, ±10 V.

See Also K_IntStart

For more information about single-mode analog output operations, refer to the following pages:

Applicable Boards	Refer to
DAS-1700ST-DA Series, DAS-1702 HR-DA	page 2-33
DAS-1700AO Series	page 2-40

Usage

The following examples show how to use the formula for converting voltages to count values for a single-mode analog output operation.

C/C++

```
#include "DASDECL.H"
...
DWORD dwDAValue;
...
dwDAValue = (DWORD) (5.0 * 4096 / 20) + 2048;
wDasErr = K_DAWriteGain (hDev, 0, 0, dwDAValue);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
dwDAValue : Longint;
...
dwDAValue := Round((5.0 * 4096.0 / 20.0) + 2048);
wDasErr := K_DAWriteGain (hDev, 0, 0, dwDAValue);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global dwDAValue As Long
...
dwDAValue = INT(5.0 * 4096! / 20!) + 2048
wDasErr = K_DAWriteGain (hDev, 0, 0, dwDAValue)
```

Boards Supported

All

Purpose

Reads a single digital input value.

Prototype

C/C++

DASErr far pascal K_DIRead (DWORD *hDev*, BYTE *nChan*, void far **pData*);

Turbo Pascal for Windows

Function K_DIRead (*hDev* : Longint; *nChan* : Byte; *pData* : Pointer) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DIRead Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nChan* As Integer, *pData* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
nChan	Digital input channel. Valid value: **0**
pData	Digital input value.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function reads the values of all digital input lines on the board specified by *hDev*, and stores the value in *pData*.

pData is a 16-bit variable. The acquired digital value is stored in bits 0, 1, 2, and 3; the values in the remaining bits of *pData* are not defined. Refer to page 2-60 for more information.

See Also

K_IntStart

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wDIValue;
...
wDasErr = K_DIRead (hDev, 0, &wDIValue);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDIValue : Word;
...
wDasErr := K_DIRead (hDev, 0, @wDIValue);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wDIValue As Integer
...
wDasErr = K_DIRead (hDev, 0, wDIValue)
```

Boards Supported

All

Purpose

Allocates a buffer for a DMA-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_DMAAlloc (DWORD *hFrame*,
DWORD *dwSamples*, void far * far **pBuf*, WORD far * *phMem*);

Turbo Pascal for Windows

Function K_DMAAlloc (*hFrame* : Longint; *dwSamples* : Longint;
pBuf : Pointer; Var *phMem* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DMAAlloc Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwSamples* As Long, *pBuf* As Long,
phMem As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

dwSamples — Number of samples.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	DMA	**1** to **65536**
Analog output	DAS-1700AO Series	DMA	**1** to **65536**
		Recycle	**1** to **2048**

pBuf — Starting address of the allocated buffer.

phMem — Handle associated with the allocated buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function allocates a buffer of the size *dwSamples* from the available memory heap. On return, *pBuf* contains the far memory address of a buffer that is suitable for a DMA-mode or recycle-mode operation and *phMem* contains the handle associated with the allocated buffer. Use **K_SetDMABuf** or **K_BufListAdd** to assign *pBuf* to the frame that defines the operation.

The data in the allocated buffer is stored as counts. Refer to Appendix B for information on converting the count value to voltage.

DAS-1700HR Series boards can store count values from 0 to 65,535. Since Visual Basic for Windows treats integers as signed numbers, values greater than 32,767 are represented as negative integers. For information on converting these values, refer to page 3-24

See Also

K_BufListAdd, K_DMAFree, K_SetDMABuf

Usage

C/C++

```
#include "DASDECL.H"
...
void far *pBuf[5];   // Pointers to allocated DMA buffer
WORD hMem[5];   // Memory Handles to buffers
...
for (i = 0; i < 5; i++) {
   wDasErr = K_DMAAlloc (hAD, 1000, &pBuf[i], &hMem[i]);
   wDasErr = K_BufListAdd (hAD, pBuf[i], 1000);
}
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : Array [0..4] of ^BufType;   { DMA buffer pointers }
hMem : Array [0..4] of Word;   { Handles to DMA buffers }
...
FOR I := 0 to 4 DO
   BEGIN
   wDasErr := K_DMAAlloc(hAD, 1000, Addr(pBuf[I]), hMem[I]);
   wDasErr := K_BufListAdd (hAD, pBuf[I], 1000);
   END;
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf(5) As Long
Global hMem(5) As Integer
...
For I% = 0 To 4
   wDasErr = K_DMAAlloc (hAD, 1000, pBuf(I%), hMem(I%))
   wDasErr = K_BufListAdd (hAD, pBuf(I%), 1000)
Next I%
```

K_DMAFree

Boards Supported

All

Purpose

Frees a buffer allocated for a DMA-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_DMAFree (WORD *hMem*);

Turbo Pascal for Windows

Function K_DMAFree (*hMem* : Word) : Integer; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DMAFree Lib "DASSHELL.DLL"
(ByVal *hMem* As Integer) As Integer

Parameters

hMem — Handle to a DMA buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the buffer specified by *hMem*. This buffer was previously allocated dynamically using **K_DMAAlloc**.

See Also

K_DMAAlloc

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_DMAFree (hMem);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_DMAFree (hMem);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_DMAFree (hMem)
```

K_DMAStart

Boards Supported

All

Purpose

Starts a DMA-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_DMAStart (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_DMAStart (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DMAStart Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function starts the DMA-mode or recycle-mode operation defined by *hFrame*.

For an analog output operation on the DAS-1700AO Series boards, if the user-defined buffer in computer memory contains 2048 values or fewer and continuous buffering mode is specified, the driver starts the operation in recycle mode. If the user-defined buffer in computer memory contains more than 2048 values or if single-cycle buffering mode is specified, the driver starts the operation in DMA mode. In both cases, you can use **K_DMAStart** to start the operation.

See Also

K_DMAStatus, K_DMAStop

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	DMA	page 2-5
Analog output	DAS-1700AO Series	DMA	page 2-41
		Recycle	page 2-42

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_DMAStart (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_DMAStart (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_DMAStart (hAD)
```

K_DMAStatus

Boards Supported

All

Purpose

Gets the status of a DMA-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_DMAStatus (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal for Windows

Function K_DMAStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DMAStatus Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

Parameters

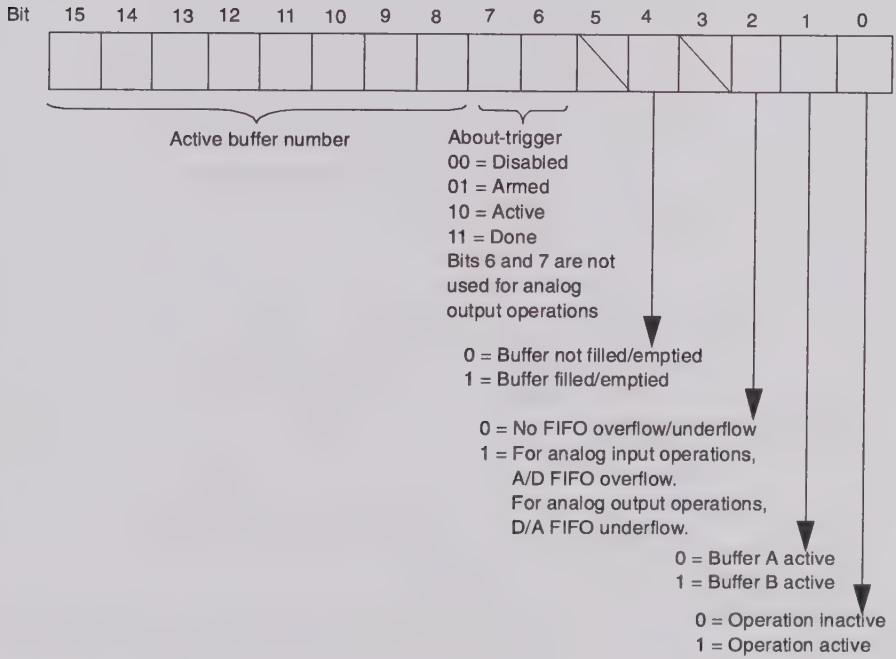
hFrame	Handle to the frame that defines the operation.
pStatus	Status of DMA-mode or recycle-mode operation; refer to **Remarks** for value stored.
pCount	For analog input operations on all DAS-1700 Series boards, number of samples that were acquired into the current buffer or starting location of post-trigger samples. Refer to **Remarks** for more information. For recycle-mode analog output operations on the DAS-1700AO Series boards, number of values that were transferred from the current buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the DMA-mode or recycle-mode operation defined by *hFrame*, this function stores the status in *pStatus*. The value stored in *pStatus* depends on the settings in the Status word, as shown below:


Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Active buffer number
About-trigger
00 = Disabled
01 = Armed
10 = Active
11 = Done
Bits 6 and 7 are not used for analog output operations
0 = Buffer not filled/emptied
1 = Buffer filled/emptied
0 = No FIFO overflow/underflow
1 = For analog input operations, A/D FIFO overflow.
For analog output operations, D/A FIFO underflow.
0 = Buffer A active
1 = Buffer B active
0 = Operation inactive
1 = Operation active

The bits are described as follows:

- Bit 0: Indicates whether a DMA-mode or recycle-mode operation is in progress.
- Bit 1: The Buffer A/B active bit. If you are using multiple buffers, this bit toggles each time a buffer is switched. If you are using a single buffer, this bit is always 0.
- Bit 2: For analog input operations, this bit indicates that the onboard A/D FIFO has overflowed. For analog output operations, this bit indicates whether the onboard D/A FIFO has underflowed. The overflow or underflow event automatically stops all conversions.
- Bit 3: Not used for DMA mode or recycle mode.

- Bit 4: This bit is used during continuous buffering mode. For analog input operations, this bit is set when all data acquisition buffers that are currently assigned to the active operation have been filled with data at least once. For analog output operations, this bit is set when all buffers that are currently assigned to the active operation have been emptied at least once.
- Bit 5: Unassigned.
- Bits 6-7: For analog input operations, these bits indicate the state of the about trigger. For analog output operations, these bits are not used.
- Bits 8-15: In multiple-buffer acquisitions, these bits indicate the active buffer number. To determine the buffer number, divide the value of the Status word by 256.

For analog input operations, the meaning of *pCount* depends on whether you are using about-trigger acquisition, as follows:

- If you are not using about-trigger acquisition (bits 6 and 7 = 00), *pCount* contains the number of samples acquired.
- If you are using about-trigger acquisition and the operation is still in progress (bit 0 = 1, bits 6 and 7 = 01 or 10), *pCount* always contains 0.
- If you are using about-trigger acquisition and the operation is complete (bit 0 = 0, bits 6 and 7 = 11), *pCount* contains the location of the first post-trigger sample.

In single-cycle mode, if the desired number of samples is acquired before the about-trigger event occurs, *pCount* contains the number of samples acquired.

If you stop the operation manually using **K_DMAStop** (whether or not the about-trigger event occurred) or if an error occurs, *pCount* contains the location of the last transferred sample.

You can use the value of *pCount* to determine which buffers contain the post-trigger samples. For example, assume that you allocated five buffers of 20,000 samples each, that *pCount* = 65,000, and that you specified 20,000 post-trigger samples in **K_SetAboutTrig**. The first post-trigger sample is located in Buffer 4, address 65,000, with post-trigger samples located in Buffer 4 and Buffer 5. (Note that if you allocated four buffers and bit 4 = 1 (continuous buffering mode), Buffer 4 and Buffer 1 contain post-trigger samples.)

For analog output operations (DAS-1700AO Series boards only), *pCount* stores the number of values transferred to the D/A FIFO.

For an analog output operation on a DAS-1700AO Series board, if the user-defined buffer in computer memory contains 2048 values or fewer and continuous buffering mode is specified, the driver assumes that the operation was started in recycle mode. If the user-defined buffer in computer memory contains more than 2048 values, or if single-cycle buffering mode is specified, the driver assumes that the operation was stared in DMA mode. In both cases, you can use the **K_DMAStatus** function to get the status of the operation.

See Also

K_DMAStart, K_DMAStop

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	DMA	page 2-5
Analog output	DAS-1700AO Series	DMA	page 2-41
		Recycle	page 2-42

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_DMAStatus (hAD, &wStatus, &dwCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStatus (hAD, wStatus, dwCount);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_DMAStatus (hAD, wStatus, dwCount)
```

Boards Supported

All

Purpose

Stops a DMA-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_DMAStop (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal for Windows

Function K_DMAStop (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DMAStop Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of DMA-mode or recycle-mode operation; refer to **Remarks** for **K_DMAStatus** on page 4-39 for the value stored.
pCount	For analog input operations (all DAS-1700 Series board), number of samples that were acquired into the current buffer or starting location of post-trigger samples. Value stored: **0** to **65536** For analog output operations (DAS-1700AO Series boards only), number of samples that were transferred from the current buffer. Refer to **Remarks** for **K_DMAStatus** on page 4-39 for more information.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function stops the DMA-mode or recycle-mode operation defined by *hFrame* and stores the status of the operation in *pStatus*. For analog input operations, *pCount* stores the number of samples that were acquired into the current buffer or the starting location of the post-trigger samples. For analog output operations (DAS-1700AO Series boards only), *pCount* store the number of samples transferred to the D/A FIFO.

If a DMA-mode or recycle-mode operation is not in progress, **K_DMAStop** is ignored.

See Also

K_DMAStart, K_DMAStatus

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	DMA	page 2-5
Analog output	DAS-1700AO Series	DMA	page 2-41
		Recycle	page 2-42

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_DMAStop (hAD, &wStatus, &dwCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_DMAStop (hAD, wStatus, dwCount);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_DMAStop (hAD, wStatus, dwCount)
```

K_DOWrite

Boards Supported

All

Purpose

Writes a single digital output value to the digital output channel.

Prototype

C/C++

DASErr far pascal K_DOWrite (DWORD *hDev*, BYTE *nChan*, DWORD *dwData*);

Turbo Pascal for Windows

Function K_DOWrite (*hDev* : Longint; *nChan* : Byte; *dwData* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_DOWrite Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nChan* As Integer, ByVal *dwData* As Long) As Integer

Parameters

hDev	Handle associated with the board.
nChan	Digital output channel. Valid value: **0**
dwData	Digital output value. Valid values: **0** to **15**

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function writes the value *dwData* to the digital output lines on the board specified by *hDev*.

dwData is a 32-bit variable. The value to be written is stored in bits 0 through 3 for the DAS-1700 Series boards; the values in the remaining bits of *dwData* are not defined. Refer to page 2-61 for more information.

See Also

K_IntStart

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD dwDOValue;
...
dwDOValue = 0x5;
wDasErr = K_DOWrite (hDev, 0, dwDOValue);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
dwDOValue : Longint;
...
dwDOValue := $5;
wDasErr := K_DOWrite (hDev, 0, dwDOValue);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global dwDOValue As Long
...
dwDOValue = &H5
wDasErr = K_DOWrite (hDev, 0, dwDOValue)
```

K_FormatChnGAry

Boards Supported

All

Purpose

Converts the format of a channel-gain queue.

Prototype

C/C++
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_FormatChnGAry Lib "DASSHELL.DLL" (*pArray* As Integer) As Integer

Parameters

pArray	Channel-gain queue starting address.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function uses double-byte (16-bit) values to convert a channel-gain queue created in Visual Basic for Windows to a channel-gain queue of single-byte (8-bit) values that the **K_SetChnGAry** function can use. **K_FormatChnGAry** then stores the starting address of the converted channel-gain queue in *pArray*.

After you use this function, your program can no longer read the converted queue. You must use the **K_RestoreChnGAry** function to return the queue to its original format. Refer to page 4-95 for more information.

See Also

K_RestoreChnGAry, K_SetChnGAry

Usage

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global ChanGainArray(16) As Integer 'Chan/Gain array
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2   ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = K_FormatChnGAry (ChanGainArray(0))
```

Boards Supported

All

Purpose

Frees a previously specified device handle.

Prototype

C/C++

DASErr far pascal K_FreeDevHandle (DWORD *hDev*);

Turbo Pascal for Windows

Function K_FreeDevHandle (*hDev* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_FreeDevHandle Lib "DASSHELL.DLL"
(ByVal *hDev* As Long) As Integer

Parameters

hDev	Device handle you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the device handle specified by *hDev,* as well as all frame handles associated with *hDev.*

See Also

K_GetDevHandle

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_FreeDevHandle (hDev);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_FreeDevHandle (hDev);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_FreeDevHandle (hDev)
```

Boards Supported

All

Purpose

Frees a frame.

Prototype

C/C++

DASErr far pascal K_FreeFrame (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_FreeFrame (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_FreeFrame Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame you want to free.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the frame specified by *hFrame*, making the frame available for another operation.

See Also

K_GetADFrame, K_GetDAFrame, K_GetDIFrame, K_GetDOFrame

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_FreeFrame (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_FreeFrame (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_FreeFrame (hAD)
```

K_GetADCommonMode

Boards Supported

All

Purpose

Gets a DAS board's analog input common-mode ground reference.

Prototype

C/C++

DASErr far pascal K_GetADCommonMode (DWORD *hDev*,
WORD far **pMode*);

Turbo Pascal for Windows

Function K_GetADCommonMode (*hDev* : Longint;
Var *pMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetADCommonMode Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pMode* As Integer) As Integer

Parameters

hDev	Handle to the associated board.
pMode	Common-mode ground reference. Value stored: **0** = LL-GND **1** = User-defined

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the code that indicates the common-mode ground reference in *pMode*.

See Also

K_SetADCommonMode

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wADCommMode;
...
wDasErr = K_GetADCommonMode (hDev, &wADCommMode);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wADCommMode : Word;
...
wDasErr := K_GetADCommonMode (hDev, wADCommMode);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wADCommMode As Integer
...
wDasErr = K_GetADCommonMode (hDev, wADCommMode)
```

K_GetADConfig

Boards Supported

All

Purpose

Gets a DAS board's analog input channel configuration.

Prototype

C/C++

DASErr far pascal K_GetADConfig (DWORD *hDev*,
WORD far **pMode*);

Turbo Pascal for Windows

Function K_GetADConfig (*hDev* : Longint; Var *pMode* : Word) : Word;
far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetADConfig Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *pMode* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
pMode	Analog input channel configuration. Value stored: **0** = Differential **1** = Single-ended

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the code that indicates the analog input channel configuration in *pMode*.

See Also

K_SetADConfig

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wADConfig;
...
wDasErr = K_GetADConfig (hDev, &wADConfig);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wADConfig : Word;
...
wDasErr := K_GetADConfig (hDev, wADConfig);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wADConfig As Integer
...
wDasErr = K_GetADConfig (hDev, wADConfig)
```

K_GetADFrame

Boards Supported

All

Purpose

Accesses an A/D frame for an analog input operation.

Prototype

C/C++

DASErr far pascal K_GetADFrame (DWORD *hDev*,
DWORD far * *phFrame*);

Turbo Pascal for Windows

Function K_GetADFrame (*hDev* : Longint;
Var *phFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetADFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *phFrame* As Long) As Integer

Parameters

hDev	Handle associated with the board.
phFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function specifies that you want to perform an interrupt-mode or DMA-mode analog input operation on the board specified by *hDev*, and accesses an available A/D frame with the handle *phFrame*. The frame is initialized to the default settings listed in Table 2-1 on page 2-8.

The value stored in *phFrame* is intended to be used exclusively as an argument to functions that require a frame handle. Your program should not modify the value stored in *phFrame*.

See Also

K_ClearFrame, K_FreeFrame

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hAD;
...
wDasErr = K_GetADFrame (hDev, &hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hAD : Longint;
...
wDasErr := K_GetADFrame (hDev, hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global hAD As Long
...
wDasErr = K_GetADFrame (hDev, hAD)
```

K_GetADMode

Boards Supported

All

Purpose

Gets a DAS board's analog input range type.

Prototype

C/C++

DASErr far pascal K_GetADMode (DWORD *hDev*, WORD far **pMode*);

Turbo Pascal for Windows

Function K_GetADMode (*hDev* : Longint; Var *pMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetADMode Lib "DASSHELL.DLL" (ByVal *hDev* As Long, *pMode* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
pMode	Analog input range type. Value stored: **0** = Bipolar **1** = Unipolar

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the code that indicates the analog input range type in *pMode*.

See Also

K_SetADMode

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wADMode;
...
wDasErr = K_GetADMode (hDev, &wADMode);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wADMode : Word;
...
wDasErr := K_GetADMode (hDev, wADMode);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wADMode AS Integer
...
wDasErr = K_GetADMode (hDev, wADMode
```

K_GetBurstTicks

Boards Supported

All

Purpose

Gets the number of clock ticks between conversions to determine the burst mode conversion rate.

Prototype

C/C++

DASErr far pascal K_GetBurstTicks (DWORD *hFrame*,
short far **pTicks*);

Turbo Pascal for Windows

Function K_GetBurstTicks (*hFrame* : Longint;
Var *pTicks* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetBurstTicks Lib "DASSHELL.DLL"
(ByVal *hFrame* as Long, *pTicks* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pTicks	Number of clock ticks between conversions.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions of each channel in a scan in *pTicks*.

The *pTicks* variable contains the value of the Burst Clock Rate element.

See Also

K_SetBurstTicks

Usage

C/C++

```
#include "DASDECL.H"
...
int nCount;
...
wDasErr = K_GetBurstTicks (hAD, &nCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
nCount : Word;
...
wDasErr := K_GetBurstTicks (hAD, nCount);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global nCount As Integer
...
wDasErr = K_GetBurstTicks (hAD, nCount)
```

K_GetClkRate

Boards Supported

All

Purpose

Gets the clock rate (number of clock ticks) for the internal pacer clock.

Prototype

C/C++

DASErr far pascal K_GetClkRate (DWORD *hFrame*,
DWORD far **pRate*);

Turbo Pascal for Windows

Function K_GetClkRate (*hFrame* : Longint; Var *pRate* : Longint) : Word;
far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pRate* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pRate	Number of clock ticks between conversions.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions in *pRate*.

The *pRate* variable contains the value of the Pacer Clock Rate element.

After an operation starts on a DAS-1700 Series board, the value stored in *pRate* represents the actual count used by the internal A/D pacer clock or the internal D/A pacer clock. The actual count is not necessarily the count set by **K_SetClkRate**.

See Also

K_SetClkRate

For more information on the internal pacer clocks, refer to the following pages:

Internal Pacer Clock	Applicable Boards	Operation	Refer to
A/D	All	Analog input	page 2-20
		Digital I/O	page 2-61
	DAS-1700ST-DA Series DAS-1702HR-DA	Analog output	page 2-37
D/A	DAS-1700AO Series	Analog output	page 2-50

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD dwRate;
...
wDasErr = K_GetClkRate (hAD, &dwRate);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
dwRate : Longint;
...
wDasErr := K_GetClkRate (hAD, dwRate);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global dwRate As Long
...
wDasErr = K_GetClkRate (hAD, dwRate)
```

K_GetDAFrame

Boards Supported
DAS-1701ST-DA, DAS-1702ST-DA, DAS-1702HR-DA, DAS-1701AO, DAS-1702AO

Purpose
Accesses a D/A frame for an analog output operation.

Prototype

C/C++
DASErr far pascal K_GetDAFrame (DWORD *hDev*,
DWORD far * *phFrame*);

Turbo Pascal for Windows
Function K_GetDAFrame (*hDev* : Longint;
Var *phFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetDAFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *phFrame* As Long) As Integer

Parameters

hDev	Handle associated with the board.
phFrame	Handle to the frame that defines the analog output operation.

Return Value
Error/status code. Refer to Appendix A.

Remarks
This function specifies that you want to perform an interrupt-mode, DMA-mode, or recycle-mode analog output operation on the board specified by *hDev* and accesses an available D/A frame with the handle *phFrame*. The frame is initialized to its default settings. The default settings for the DAS-1700ST-DA Series and DAS-1702HR-DA boards are listed in Table 2-3 on page 2-34; the default settings for the DAS-1700AO Series boards are listed in Table 2-4 on page 2-43.

The value stored in *phFrame* is intended to be used exclusively as an argument to functions that require a frame handle. Your program should not modify the value stored in *phFrame*.

See Also
K_FreeFrame, K_ClearFrame

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hDA;
...
wDasErr = K_GetDAFrame (hDev, &hDA);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hDA : Longint;
...
wDasErr := K_GetDAFrame (hDev, hDA);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global hDA As Long
...
wDasErr = K_GetDAFrame (hDev, hDA)
```

K_GetDevHandle

Boards Supported

All

Purpose

Initializes any Keithley DAS board.

Prototype

C/C++

DASErr far pascal K_GetDevHandle (DWORD *hDrv*,
WORD *nBoardNum*, DWORD far * *phDev*);

Turbo Pascal for Windows

Function K_GetDevHandle (*hDrv* : Longint; *nBoardNum* : Integer;
Var *phDev* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetDevHandle Lib "DASSHELL.DLL"
(ByVal *hDrv* As Long, ByVal *nBoardNum* As Integer, *phDev* As Long)
As Integer

Parameters

hDrv	Driver handle of the associated Function Call Driver.
nBoardNum	Board number. Valid values: **0** to **2**
phDev	Handle associated with the board.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function initializes the board associated with *hDrv* and specified by *nBoardNum*, and stores the device handle of the specified board in *phDev*.

The value stored in *phDev* is intended to be used exclusively as an argument to functions that require a device handle. Your program should not modify the value stored in *phDev*.

See Also

K_FreeDevHandle

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hDev;
...
wDasErr = K_GetDevHandle (hDrv, 0, &hDev);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hDev : Longint;
...
wDasErr := K_GetDevHandle (hDrv, 0, hDev);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global hDev As Long
...
wDasErr = K_GetDevHandle (hDrv, 0, hDev)
```

K_GetDIFrame

Boards Supported

All

Purpose

Accesses a DI frame for a digital input operation.

Prototype

C/C++

DASErr far pascal K_GetDIFrame (DWORD *hDev*,
DWORD far * *phFrame*);

Turbo Pascal for Windows

Function K_GetDIFrame (*hDev* : Longint;
Var *phFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetDIFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *phFrame* As Long) As Integer

Parameters

hDev	Handle associated with the board.
phFrame	Handle to the frame that defines the digital input operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function specifies that you want to perform an interrupt-mode digital input operation on the board specified by *hDev*, and accesses an available DI frame with the handle *phFrame*. The frame is initialized to its default settings; the default settings are listed in Table 2-6 on page 2-58.

The value stored in *phFrame* is intended to be used exclusively as an argument to functions that require a frame handle. Your program should not modify the value stored in *phFrame*.

See Also K_FreeFrame, K_ClearFrame

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hDI;
...
wDasErr = K_GetDIFrame (hDev, &hDI);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hDI : Longint;
...
wDasErr := K_GetDIFrame (hDev, hDI);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global hDI As Long
...
wDasErr = K_GetDIFrame (hDev, hDI)
```

K_GetDOFrame

Boards Supported

All

Purpose

Accesses a DO frame for a digital output operation.

Prototype

C/C++

DASErr far pascal K_GetDOFrame (DWORD *hDev*,
DWORD far * *phFrame*);

Turbo Pascal for Windows

Function K_GetDOFrame (*hDev* : Longint;
Var *phFrame* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetDOFrame Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, *phFrame* As Long) As Integer

Parameters

hDev	Handle associated with the board.
phFrame	Handle to the frame that defines the digital output operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function specifies that you want to perform an interrupt-mode digital output operation on the board specified by *hDev* and accesses an available DO frame with the handle *phFrame*. The frame is initialized to its default settings; the default settings are listed in Table 2-7 on page 2-59.

The value stored in *phFrame* is intended to be used exclusively as an argument to functions that require a frame handle. Your program should not modify the value stored in *phFrame*.

See Also

K_FreeFrame, K_ClearFrame

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hDO;
...
wDasErr = K_GetDOFrame (hDev, &hDO);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
hDO : Longint;
...
wDasErr := K_GetDOFrame (hDev, hDO);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global hDO As Long
...
wDasErr = K_GetDOFrame (hDev, hDO)
```

K_GetErrMsg

Boards Supported

All

Purpose

Gets the address of an error message string.

Prototype

C/C++

DASErr far pascal K_GetErrMsg (DWORD *hDev*, short *nDASErr*, char far * far * *pErrMsg*);

Turbo Pascal for Windows

Not supported

Visual Basic for Windows

Not supported

Parameters

hDev	Handle associated with the board.
nDASErr	Error message number.
pErrMsg	Address of error message string.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the address of the string corresponding to error message number *nDASErr* in *pErrMsg*.

Refer to page 2-3 for more information about error handling. Refer to Appendix A for a list of error codes and their meanings.

Usage

C/C++

```
#include "DASDECL.H"
...
char far *pErrMsg;
...
Error = K_GetErrMsg (hDev, wDasErr, &pErrMsg);
```

Boards Supported All

Purpose Gets the current DAS shell version.

Prototype

C/C++
DASErr far pascal K_GetShellVer (WORD far **pVersion*);

Turbo Pascal for Windows
Function K_GetShellVer (Var *pVersion* : Word) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_GetShellVer Lib "DASSHELL.DLL"
(*pVersion* As Integer) As Integer

Parameters

pVersion	A word value containing the major and minor version numbers of the DAS shell.

Return Value Error/status code. Refer to Appendix A.

Remarks This function stores the current DAS shell version in *pVersion*.

To obtain the major version number of the DAS shell, divide *pVersion* by 256. To obtain the minor version number of the DAS shell, perform a Boolean AND operation with *pVersion* and 255 (0FF hex).

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wShellVer;
...
wDasErr = K_GetShellVer (&wShellVer);
printf ("Shell Ver %d.%d", wShellVer >> 8, wShellVer & 0xff);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wShellVer : Word;
...
wDasErr := K_GetShellVer (wShellVer);
FormatStr(VerStr,' %4x ', nShellVer / 256, '.', nShellVer And $ff);
writeln (' Shell Ver ', VerStr);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wShellVer As Integer
...
wDasErr = K_GetShellVer (wShellVer)
ShellVer$ = LTRIM$ (STR$(INT(wShellVer / 256))) + "." +
   LTRIM$ (STR$(wShellVer AND &HFF))
MsgBox "Shell Version: " + ShellVer$
```

Boards Supported

All

Purpose

Gets revision numbers.

Prototype

C/C++

DASErr far pascal K_GetVer (DWORD *hDev*, short far * *pSpecVer*, short far * *pDrvVer*);

Turbo Pascal for Windows

Function K_GetVer (*hDev* : Longint; Var *pSpecVer* : Word; Var *pDrvVer* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_GetVer Lib "DASSHELL.DLL" (ByVal *hDev* As Long, *pSpecVer* As Integer, *pDrvVer* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
pSpecVer	Revision number of the Keithley DAS Driver Specification to which the driver conforms.
pDrvVer	Driver version number.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function stores the revision number of the DAS-1700 Series Function Call Driver in *pDrvVer* and the revision number of the driver specification in *pSpecVer*.

The values stored in *pSpecVer* and *pDrvVer* are two-byte (16-bit) integers; the high byte of each contains the major revision level and the low byte of each contains the minor revision level. For example, if the driver version number is 2.10, the major revision level is 2 and the minor revision level is 10; therefore, the high byte of *pDrvVer* contains the value of **2** (512) and the low byte of *pDrvVer* contains the value of **10**; the value of both bytes is 522.

To obtain the major version number of the Function Call Driver, divide *pDrvVer* by 256; to obtain the minor version number of the Function Call Driver, perform a Boolean AND operation with *pDrvVer* and 255 (0FFh).

To obtain the major version number of the driver specification, divide *pSpecVer* by 256; to obtain the minor version number of the driver specification, perform a Boolean AND operation with *pSpecVer* and 255 (0FFh).

Usage

C/C++

```
#include "DASDECL.H"
...
short nSpecVer, nDrvVer;
...
wDasErr = K_GetVer (hDev, &nSpecVer, &nDrvVer);
printf ("Driver Ver %d.%d", nDrvVer >> 8, nDrvVer & 0xff);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
nSpecVer : Word;
nDrvVer : Word;
...
wDasErr := K_GetVer (hDev, nSpecVer, nDrvVer);
FormatStr(VerStr, ' %4x ', nDrvVer / 256, '.', nDrvVer And $ff);
writeln (' Driver Ver ', VerStr);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global nSpecVer As Integer
Global nDrvVer As Integer
...
wDasErr = K_GetVer (hDev, nSpecVer, nDrvVer)
DrvVer$ = LTRIM$ (STR$(INT(nDrvVer / 256))) + "." +
   LTRIM$ (STR$(nDrvVer AND &HFF))
MsgBox "Driver Ver: " + DrvVer$
```

Boards Supported All

Purpose Allocates a buffer for an interrupt-mode or recycle-mode operation.

Prototype

C/C++
DASErr far pascal K_IntAlloc (DWORD *hFrame*, DWORD *dwSamples*, void far * far **pBuf*, WORD far **phMem*);

Turbo Pascal for Windows
Function K_IntAlloc (*hFrame* : Longint; *dwSamples* : Longint; *pBuf* : Pointer; Var *phMem* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_IntAlloc Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *dwSamples* As Long, *pBuf* As Long, *phMem* As Integer) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

dwSamples Number of samples.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	Interrupt	1 to 5000000
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	1 to 5000000
	DAS-1700AO Series	Interrupt	1 to 5000000
		Recycle	1 to 2048
Digital I/O	All	Interrupt	1 to 32767

pBuf Starting address of the allocated buffer.

phMem Handle associated with the allocated buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function allocates a buffer of the size specified by *dwSamples*, and stores the starting address of the buffer in *pBuf* and the handle of the buffer in *phMem*.

The value stored in *phMem* is intended to be used exclusively as an argument to functions that require a memory handle. Your program should not modify the value stored in *phMem*.

For analog input and analog output operations, the data in the allocated buffer is stored as counts. Refer to Appendix B for information on converting a count value to voltage (for analog input operations) or for converting a voltage value to a count (for analog output operations).

DAS-1700HR Series boards store count values from 0 to 65,535. Since Visual Basic for Windows treats integers as signed numbers, values greater than 32,767 are represented as negative integers. For information on converting these values, refer to page 3-24.

See Also

K_BufListAdd, K_IntFree, K_SetBuf

Usage

C/C++

```
#include "DASDECL.H"
...
void far *pBuf;    // Pointer to allocated buffer
WORD hMem;    // Memory Handle to buffer
...
wDasErr = K_IntAlloc (hAD, 1000, &pBuf, &hMem);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    { buffer pointer }
hMem : Word;    { Handle to buffer }
...
wDasErr := K_IntAlloc (hAD, 1000, Addr(pBuf), hMem);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf As Long
Global hMem As Integer
...
wDasErr = K_IntAlloc (hAD, 1000, pBuf, hMem)
```

K_IntFree

Boards Supported

All

Purpose

Frees a buffer allocated for an interrupt-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_IntFree (WORD *hMem*);

Turbo Pascal for Windows

Function K_IntFree (*hMem* : Word) : Integer; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_IntFree Lib "DASSHELL.DLL"
(ByVal *hMem* As Integer) As Integer

Parameters

hMem Handle to interrupt buffer.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function frees the buffer specified by *hMem*. The buffer was previously allocated dynamically using **K_IntAlloc**.

See Also

K_IntAlloc

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_IntFree (hMem);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_IntFree (hMem);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_IntFree (hMem)
```

Boards Supported

All

Purpose

Starts an interrupt-mode or recycle-mode operation.

Prototype

C/C++
DASErr far pascal K_IntStart (DWORD *hFrame*);

Turbo Pascal for Windows
Function K_IntStart (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows
Declare Function K_IntStart Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function starts the interrupt-mode or recycle-mode operation defined by *hFrame*.

For an analog output operation on the DAS-1700AO Series boards, if the user-defined buffer in computer memory contains 2048 values or fewer and continuous buffering mode is specified, the driver starts the operation in recycle mode. If the user-defined buffer in computer memory contains more than 2048 values or if single-cycle buffering mode is specified, the driver starts the operation in interrupt mode. In both cases, you can use **K_IntStart** to start the operation.

See Also K_IntStatus, K_IntStop

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	Interrupt	page 2-5
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	page 2-33
	DAS-1700AO Series	Interrupt	page 2-40
		Recycle	page 2-42
Digital I/O	All	Interrupt	page 2-57

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_IntStart (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_IntStart (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_IntStart (hAD)
```

Boards Supported

All

Purpose

Gets the status of an interrupt-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_IntStatus (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal for Windows

Function K_IntStatus (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_IntStatus Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of interrupt-mode or recycle-mode operation; refer to **Remarks** for value stored.
pCount	Current number of samples transferred.

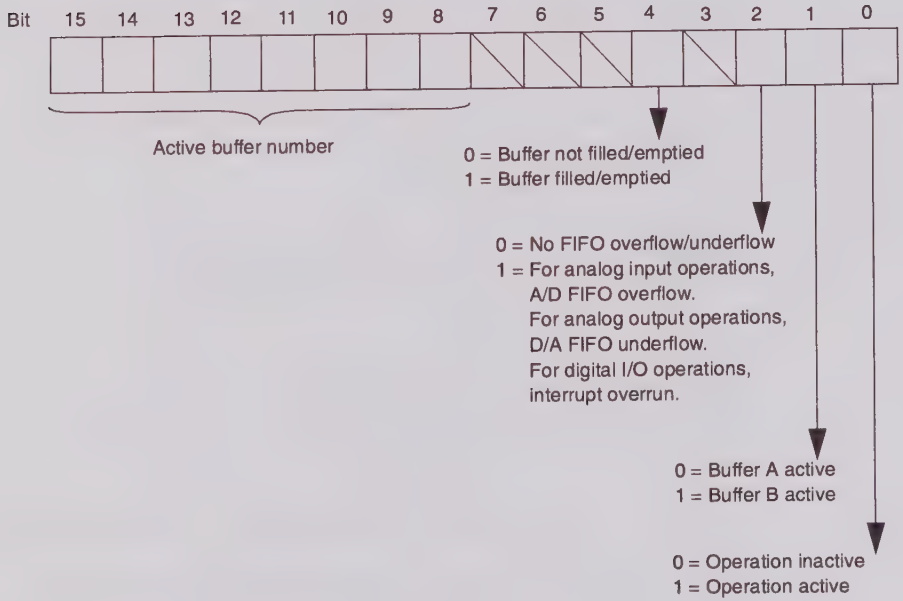
Return Value

Error/status code. Refer to Appendix A.

Remarks

For the interrupt-mode or recycle-mode operation defined by *hFrame*, this function stores the status in *pStatus* and the number of samples in *pCount*.

The value stored in *pStatus* depends on the settings in the Status word, as shown below:


Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Active buffer number
0 = Buffer not filled/emptied
1 = Buffer filled/emptied
0 = No FIFO overflow/underflow
1 = For analog input operations,
A/D FIFO overflow.
For analog output operations,
D/A FIFO underflow.
For digital I/O operations,
interrupt overrun.
0 = Buffer A active
1 = Buffer B active
0 = Operation inactive
1 = Operation active

The bits are described as follows:

- Bit 0: Indicates whether an interrupt-mode or recycle-mode operation is in progress.
- Bit 1: The Buffer A/B active bit. If you are using multiple buffers, this bit toggles each time a buffer is switched. If you are using a single buffer, this bit is always 0.
- Bit 2: For analog input operations on all DAS-1700 Series boards, this bit indicates that the onboard A/D FIFO overflowed. For analog output operations on DAS-1700AO Series boards, this bit indicates that the onboard D/A FIFO underflowed. The overflow or underflow event automatically stops all conversions. For digital I/O operations on all DAS-1700 Series boards, this bit indicates whether the board issued an interrupt while the CPU was processing a previous interrupt from the same board.

- Bit 3: Reserved.
- Bit 4: This bit is used during continuous buffering mode. For analog input and digital input operations, this bit is set when all buffers that are currently assigned to the active operation have been filled with data at least once. For analog output and digital output operations, this bit is set when all buffers that are currently assigned to the active operation have been emptied at least once.
- Bits 5-7: Unassigned.
- Bits 8-15: In multiple-buffer operations, these bits indicate the current active buffer number. To determine the active buffer number, divide the value of the Status word by 256.

For analog input and digital input operations, *pCount* stores the number of samples acquired into the current buffer. For analog output and digital output operations, *pCount* stores the number of values transferred from the current buffer.

For analog output operations on the DAS-1700AO Series boards, if the user-defined buffer in computer memory contains 2048 values or fewer and continuous buffering mode is specified, the driver assumes that the operation was started in recycle mode. If the user-defined buffer in computer memory contains more than 2048 values or if single-cycle buffering mode is specified, the driver assumes that the operation was started in interrupt mode. In both cases, you can use **K_IntStatus** to get the status of the operation.

See Also K_IntStart, K_IntStop

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	Interrupt	page 2-5
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	page 2-33
	DAS-1700AO Series	Interrupt	page 2-40
		Recycle	page 2-42
Digital I/O	All	Interrupt	page 2-57

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_IntStatus (hAD, &wStatus, &dwCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStatus (hAD, wStatus, dwCount);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_IntStatus (hAD, wStatus, dwCount)
```

Boards Supported

All

Purpose

Stops an interrupt-mode or recycle-mode operation.

Prototype

C/C++

DASErr far pascal K_IntStop (DWORD *hFrame*, short far **pStatus*, DWORD far **pCount*);

Turbo Pascal for Windows

Function K_IntStop (*hFrame* : Longint; Var *pStatus* : Word; Var *pCount* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_IntStop Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, *pStatus* As Integer, *pCount* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pStatus	Status of interrupt-mode operation; see **Remarks** for **K_IntStatus** on page 4-83 for value stored.
pCount	Current number of samples transferred.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function stops the interrupt-mode or recycle-mode operation defined by *hFrame* and stores the status of the operation in *pStatus* and the number of samples in *pCount*.

For an analog output operation on the DAS-1700AO Series boards, if the user-defined buffer in computer memory contains 2048 values or fewer and continuous buffering mode is specified, the driver assumes that the opeartion was started in recycle mode. If the user-defined buffer in computer memory contains more than 2048 values or if single-cycle buffering mode is specified, the driver assumes that the operation was started in interrupt mode. In both cases, you can use **K_IntStop** to stop the operation.

If an interrupt-mode or recycle-mode operation is not in progress, **K_IntStop** is ignored.

See Also

K_IntStart, K_IntStatus

Operation	Applicable Boards	Operation Mode	Refer to
Analog input	All	Interrupt	page 2-5
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	page 2-33
	DAS-1700AO Series	Interrupt	page 2-40
		Recycle	page 2-42
Digital I/O	All	Interrupt	page 2-57

Usage

C/C++

```
#include "DASDECL.H"
...
WORD wStatus;
DWORD dwCount;
...
wDasErr = K_IntStop (hAD, &wStatus, &dwCount);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wStatus : Word;
dwCount : Longint;
...
wDasErr := K_IntStop (hAD, wStatus, dwCount);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global wStatus As Integer
Global dwCount As Long
...
wDasErr = K_IntStop (hAD, wStatus, dwCount)
```

Boards Supported

All

Purpose

Transfers data from the program's local array to a buffer allocated through **K_IntAlloc** or **K_DMAAlloc**.

Prototype

C/C++
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_MoveArrayToBuf Lib "DASSHELL.DLL" Alias "K_MoveDataBuf" (ByVal *pDest* As Long, *pSource* As Integer, ByVal *nCount* As Integer) As Integer

Parameters

pDest	Address of destination buffer.
pSource	Address of source array.
nCount	Number of values to transfer. Valid values: **1** to **32767**

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function transfers the number of values specified by *nCount* from the array at address *pSource* to the buffer at address *pDest*.

If the buffer used to store output data for your program was allocated through **K_IntAlloc** or **K_DMAAlloc**, the buffer is not accessible to the program and you must use this function to move the data from the program's local array to the dynamically allocated buffer. If the array used to store output data for your program was dimensioned locally within the program's memory area, the array is accessible to the program and you do not have to use this function.

See Also

K_DMAAlloc, K_IntAlloc

Usage

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Dim DACArray(2000) As Integer
...
wDasErr = K_IntAlloc (hDA, 1000, pBuf, hMem)
...
wDasErr = K_MoveArrayToBuf (pBuf, DACArray(0), 1000)
```

Boards Supported

All

Purpose

Transfers data from a buffer allocated through **K_IntAlloc** or **K_DMAAlloc** to the program's local array.

Prototype

C/C++
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_MoveBufToArray Lib "DASSHELL.DLL" Alias "K_MoveDataBuf" (*pDest* As Integer, ByVal *pSource* As Long, ByVal *nCount* As Integer) As Integer

Parameters

pDest	Address of destination array.
pSource	Address of source buffer.
nCount	Number of samples to transfer. Valid values: **1** to **32767**

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function transfers the number of samples specified by *nCount* from the buffer at address *pSource* to the array at address *pDest*.

If the buffer used to store acquired data for your program was allocated through **K_IntAlloc** or **K_DMAAlloc**, the data in the buffer is not accessible to your program. You must use this function to move the data from the allocated buffer to the program's local array. If the array used to store acquired data for your program was dimensioned locally within the program's memory area, the array is accessible to your program and you do not have to use this function.

See Also

K_DMAAlloc, K_IntAlloc

Usage

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Dim ADArray(2000) As Integer
...
wDasErr = K_IntAlloc (hAD, 1000, pBuf, hMem)
...
wDasErr = K_MoveBufToArray (ADArray(0), pBuf, 1000)
```

Boards Supported

All

Purpose

Initializes any Keithley DAS Function Call Driver.

Prototype

C/C++

DASErr far pascal K_OpenDriver (char far * *szDrvName*, char far * *szCfgName*, DWORD far * *phDrv*);

Turbo Pascal for Windows

Function K_OpenDriver (Var *szDrvName* : char; Var *szCfgName* : char; Var *phDrv* : LongInt) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_OpenDriver Lib "DASSHELL.DLL" (ByVal *szDrvName* As String, ByVal *szCfgName* As String, *phDrv* As Long) As Integer

Parameters

szDrvName	Driver name. Valid value: **"DAS1700"** (for all DAS-1700 Series boards)
szCfgName	Driver configuration file. Valid values: The name of a configuration file **0** if driver has already been opened
phDrv	Handle associated with the driver.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function initializes the DAS-1700 Series Function Call Driver according to the information in the configuration file specified by *szCfgName*, and stores the driver handle in *phDrv*.

You can use this function to initialize the Function Call Driver associated with any Keithley MetraByte DAS board. For DAS-1700 Series boards, the string stored in *szDrvName* must be DAS1700. Refer to other

Function Call Driver user's guides for the appropriate string to store in *szDrvName* for other Keithley MetraByte DAS boards.

The value stored in *phDrv* is intended to be used exclusively as an argument to functions that require a driver handle. Your program should not modify the value stored in *phDrv*.

You create a configuration file using the configuration utility (CFG1700.EXE). Refer to the *DAS-1700 Series User's Guide* for more information.

If *szCfgName* = 0, **K_OpenDriver** checks whether the driver has already been opened and linked to a configuration file and, if it has, uses the current configuration.

See Also K_CloseDriver

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD hDrv;
...
wDasErr = K_OpenDriver ("DAS1700", "DAS1702.CFG", &hDrv);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
szDrvName : String;
szCfgName : String;
hDrv : Longint;
...
szDrvName := 'DAS1700' + #0;
szCfgName := 'DAS1702.CFG' + #0;
wDasErr := K_OpenDriver (szDrvName[1], szCfgName[1], hDrv);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
DIM hDrv As Long
...
wDasErr = K_OpenDriver ("DAS1700", "DAS1702.CFG", hDrv)
```

Boards Supported

All

Purpose

Restores a converted channel-gain queue.

Prototype

C/C++
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_RestoreChnGAry Lib "DASSHELL.DLL"
(*pArray* As Integer) As Integer

Parameters

pArray	Channel-gain queue starting address.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function restores the channel-gain queue at the address specified by *pArray* to its original format so that it can be used by your Visual Basic for Windows program. The channel-gain queue was converted using **K_FormatChnGAry**.

Refer to page 4-48 for more information about the **K_FormatChnGAry** function.

See Also

K_FormatChnGAry, K_SetChnGAry

Usage

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global ChanGainArray(16) As Integer ' Chan/Gain array
...
wDasErr = K_RestoreChnGAry (ChanGainArray(0))
```

K_SetAboutTrig

Boards Supported

All

Purpose

Enables the about trigger and specifies the number of post-trigger samples.

Prototype

C/C++

DASErr far pascal K_SetAboutTrig (DWORD *hFrame*,
DWORD *dwSamples*);

Turbo Pascal for Windows

Function K_SetAboutTrig (*hFrame* : Longint;
dwSamples : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetAboutTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwSamples* As Long) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
dwSamples	Number of post-trigger samples. Valid values: **1** to **65536**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the DMA-mode analog input operation defined by *hFrame*, this function enables the about trigger and specifies the number of post-trigger samples in *dwSamples*.

See Also

K_ClrAboutTrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetAboutTrig (hAD, 100);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetAboutTrig (hAD, 100);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetAboutTrig (hAD, 100)
```

K_SetADCommonMode

Boards Supported

All

Purpose

Sets a DAS board's analog input common-mode ground reference.

Prototype

C/C++

DASErr far pascal K_SetADCommonMode (DWORD *hDev*, WORD *nMode*);

Turbo Pascal for Windows

Function K_SetADCommonMode (*hDev* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetADCommonMode Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
nMode	Common-mode ground reference. Valid values: **0** = LL-GND **1** = User-defined

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function specifies the common-mode ground reference in *nMode*.

See Also

K_GetADCommonMode

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetADCommonMode (hDev, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr = K_SetADCommonMode (hDev, 0);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetADCommonMode (hDev, 0)
```

K_SetADConfig

Boards Supported

All

Purpose

Sets a DAS board's analog input channel configuration.

Prototype

C/C++

DASErr far pascal K_SetADConfig (DWORD *hDev*, WORD *nMode*);

Turbo Pascal for Windows

Function K_SetADConfig (*hDev* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetADConfig Lib "DASSHELL.DLL" (ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
nMode	Analog input channel configuration. Valid values: **0** = Differential **1** = Single-ended

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function specifies the analog input channel configuration in *nMode*.

See Also

K_GetADConfig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetADConfig (hDev, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr = K_SetADConfig (hDev, 0);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetADConfig (hDev, 0)
```

K_SetADFreeRun

Boards Supported

All

Purpose

Specifies burst conversion mode.

Prototype

C/C++

DASErr far pascal K_SetADFreeRun (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_SetADFreeRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetADFreeRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

This function sets the conversion mode for the operation defined by *hFrame* to burst mode. Refer to page 2-18 for information on conversion modes.

See Also

K_ClrADFreeRun

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetADFreeRun (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetADFreeRun (hAD);
```

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetADFreeRun (hAD)
```

K_SetADMode

Boards Supported

All

Purpose

Sets the analog input range type for a DAS board.

Prototype

C/C++

DASErr far pascal K_SetADMode (DWORD *hDev*, WORD *nMode*);

Turbo Pascal for Windows

Function K_SetADMode (*hDev* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetADMode Lib "DASSHELL.DLL"
(ByVal *hDev* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hDev	Handle associated with the board.
nMode	Analog input range type. Valid values: **0** = Bipolar **1** = Unipolar

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the board specified by *hDev*, this function specifies the analog input range type in *nMode*.

See Also

K_GetADMode

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetADMode (hDev, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr = K_SetADMode (hDev, 0);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetADMode (hDev, 0)
```

K_SetADTrig

Boards Supported

All

Purpose

Sets up an analog start trigger.

Prototype

C/C++

DASErr far pascal K_SetADTrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *dwLevel*);

Turbo Pascal for Windows

Function K_SetADTrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *dwLevel* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetADTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nOpt* As Integer,
ByVal *nChan* As Integer, ByVal *dwLevel* As Long) As Integer

Parameters

hFrame

Handle to the frame that defines the operation.

nOpt

Analog trigger polarity and sensitivity.
Valid values: **0** = Positive edge
2 = Negative edge

nChan

Analog input channel used as trigger channel.
Valid values:

No EXP-1800 expansion accessories attached	Differential: **0** to **7** Single-ended: **0** to **15**
n EXP-1800 expansion accessories attached	**0** to **15**(***n*** + **1**)

dwLevel

Level at which the trigger event occurs.
Valid values:

DAS-1700ST Series DAS-1700AO Series	Unipolar: **0** to **4095** Bipolar: –**2048** to **2047**
DAS-1700HR Series	Unipolar: **0** to **65535** Bipolar: –**32768** to **32767**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the channel used for an analog trigger in *nChan*, the level used for the analog trigger in *dwLevel*, and the trigger polarity and trigger sensitivity in *nOpt*.

You specify the value for *dwLevel* in counts. Refer to Appendix B for information on converting the actual voltage to a count value.

The *nOpt* variable sets the value of the Trigger Polarity and Trigger Sensitivity elements; the *nChan* variable sets the value of the Trigger Channel element; the *dwLevel* variable sets the value of the Trigger Level element.

For DAS-1700HR Series boards, the count value for *nHyst* can be as high as 65,535. Since Visual Basic for Windows treats integers as signed numbers, you must represent values greater than 32,767 as negative integers. For information on converting these values, refer to page 3-24.

K_SetADTrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

See Also

K_SetTrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetADTrig (hAD, 0, 0, 2047);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetADTrig (hAD, 0, 0, 2047);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetADTrig (hAD, 0, 0, 2047)
```

K_SetBuf

Boards Supported

All

Purpose

Specifies the starting address of a previously allocated or dimensioned buffer or array and the number of samples in the buffer or array.

Prototype

C/C++

DASErr far pascal K_SetBuf (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal for Windows

Function K_SetBuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetBuf Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *pBuf* As Long, ByVal *dwSamples* As Long) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

pBuf — Starting address of the buffer or array.

dwSamples — Number of samples.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	Interrupt	**1** to **5000000**
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	**1** to **5000000**
	DAS-1700AO Series	Interrupt	**1** to **5000000**
		Recycle	**1** to **2048**
Digital I/O	All	Interrupt	**1** to **32767**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the starting address of a previously allocated or dimensioned buffer or array in *pBuf*, and the number of samples in the buffer or array in *dwSamples*.

For C/C++ and Pascal programs, use this function whether you dimensioned your array locally or allocated your buffer dynamically using **K_IntAlloc**. For buffers allocated dynamically using **K_DMAAlloc**, use **K_SetDMABuf**.

For Visual Basic for Windows, use this function only for buffers allocated dynamically using **K_IntAlloc**. For buffers allocated dynamically using **K_DMAAlloc**, use **K_SetDMABuf**. For locally dimensioned arrays, use **K_SetBufI**.

Do not use this function if you are using multiple buffers. Use **K_BufListAdd** to specify the starting addresses of multiple buffers.

The *pBuf* variable sets the value of the Buffer element; the *dwSamples* variable sets the value of the Number of Samples element.

See Also

K_IntAlloc, K_BufListAdd, K_SetBufI

Usage

C/C++

```
#include "DASDECL.H"
...
void far *pBuf;   // Pointer to allocated buffer
...
wDasErr = K_IntAlloc (hAD, 1000, &pBuf, &hMem);
wDasErr = K_SetBuf (hAD, pBuf, 1000);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;   { buffer pointer }
...
wDasErr := K_IntAlloc (hAD, 1000, Addr(pBuf), hMem);
wDasErr := K_SetBuf (hAD, pBuf, 1000);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf As Long
...
wDasErr = K_IntAlloc (hAD, 1000, pBuf, hMem)
wDasErr = K_SetBuf (hAD, pBuf, 1000)
```

Boards Supported

All

Purpose

Specifies the starting address of a locally dimensioned integer array and the number of samples in the array.

Prototype

C/C++
Not supported

Turbo Pascal for Windows
Not supported

Visual Basic for Windows
Declare Function K_SetBufI Lib "DASSHELL.DLL" Alias "K_SetBuf" (ByVal *hFrame* As Long, *pBuf* As Integer, ByVal *dwSize* As Long) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

pBuf — Starting address of the locally dimensioned integer array.

dwSize — Number of samples.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	Interrupt	**1** to **32767**
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	Interrupt	**1** to **32767**
	DAS-1700AO Series	Interrupt	**1** to **32767**
		Recycle	**1** to **2048**
Digital I/O	All	Interrupt	**1** to **32767**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the starting address of a locally dimensioned integer array in *pBuf* and the number of samples stored in the array in *dwSize*.

Do not use this function for C/C++ and Pascal; for these languages, use **K_SetBuf**.

For Visual Basic for Windows, use this function only for locally dimensioned arrays. For buffers allocated dynamically using **K_IntAlloc**, use **K_SetBuf**. For buffers allocated dynamically using **K_DMAAlloc**, use **K_SetDMABuf**.

Do not use this function if you are using multiple buffers. Instead, use **K_BufListAdd** to specify the starting addresses of multiple buffers.

The *pBuf* variable sets the value of the Buffer element; the *dwSize* variable sets the value of the Number of Samples element.

See Also

K_IntAlloc, K_BufListAdd, K_SetBuf

Usage

Visual Basic for Windows
(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global ADData(2000) As Integer
...
wDasErr = K_SetBufI (hAD, ADData(0), 2000)
```

Boards Supported

All

Purpose

Sets the burst mode conversion rate.

Prototype

C/C++

DASErr far pascal K_SetBurstTicks (DWORD *hFrame*, short *nTicks*);

Turbo Pascal for Windows

Function K_SetBurstTicks (*hFrame* : Longint; *nTicks* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetBurstTicks Lib "DASSHELL.DLL" (ByVal hFrame as Long, ByVal *nTicks* As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nTicks — The number of clock ticks between conversions of each channel in a scan.
Valid values:

DAS-1700ST Series DAS-1700AO Series	**6** to **64**
DAS-1700HR Series	**20** to **64**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function stores the number of clock ticks between conversions in *nTicks*.

If *nTicks* is defined as zero or less than zero, the driver assumes the minimum number of burst ticks for the related board. Refer to the *DAS-1700 Series User's Guide* for more information.

Refer to page 2-22 for more information on burst mode conversion rate.

See Also

K_GetBurstTicks

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetBurstTicks (hAD, 20);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetBurstTicks (hAD, 20);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetBurstTicks (hAD, 20)
```

Boards Supported

All

Purpose

Specifies a single channel.

Prototype

C/C++

DASErr far pascal K_SetChn (DWORD *hFrame*, short *nChan*);

Turbo Pascal for Windows

Function K_SetChn (*hFrame* : Longint; *nChan* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetChn Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nChan* As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nChan — Channel on which to perform operation. Valid values:

Operation	Applicable Boards	Channel Numbers
Analog input	All, with no EXP-1800 expansion accessories attached	Differential: **0** to **7** Single-ended: **0** to **15**
	All, with *n* EXP-1800 expansion accessories attached	**0** to **15(*n* + 1)**
Analog output	DAS-1700ST-DA Series	**0** to **3**
	DAS-1700AO Series DAS-1702HR-DA	**0** to **1**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the single channel used in *nChan*.

The *nChan* variable sets the value of the Start Channel element and the Stop Channel element.

See Also

K_SetStartStopChan

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetChn (hAD, 2);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetChn (hAD, 2);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetChn (hAD, 2)
```

Boards Supported

All

Purpose

Specifies the starting address of a channel-gain queue.

Prototype

C/C++

DASErr far pascal K_SetChnGAry (DWORD *hFrame*, void far **pArray*);

Turbo Pascal for Windows

Function K_SetChnGAry (*hFrame* : Longint;
Var *pArray* : Integer) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetChnGAry Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, *pArray* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
pArray	Channel-gain queue starting address.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the starting address of the channel-gain queue in *pArray*.

The *pArray* variable sets the value of the Channel-Gain Queue element.

If you created your channel-gain queue in Visual Basic for Windows, you must use **K_FormatChnGAry** to convert the channel-gain queue before you specify the address with **K_SetChnGAry**. Refer to page 4-48 for more information on converting the channel-gain queue.

See Also

K_FormatChnGAry, K_RestoreChnGAry

For more information about setting up a channel-gain queue, refer to the following pages:

Operation	Applicable Boards	Refer to
Analog input	All	page 2-17
Analog output	DAS-1700AO Series	page 2-49

Usage

C/C++

```
#include "DASDECL.H"
...
// DECLARE AND INITIALIZE CHAN/GAIN PAIRS
// (GainChanTable-TYPE IS DEFINED IN dasdecl.h)
GainChanTable ChanGainArray= {2,   // # of entries
   0, 0,    // chan 0, gain 1
   1, 1};   // chan 1, gain 2 (DAS-1702)
...
wDasErr = K_SetChnGAry (hAD, &ChanGainArray);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
{ Define Gain/Channel array type }
TYPE GainChanTable = Record
       num_of_codes : Integer;
       queue : Array[0..15] of Byte;
       END;
CONST ChanGainArray : GainChanTable = (
       num_of_codes : (8);   { # of chan/gain pairs }
       queue : (0,0, 1,1)
);
...
wDasErr := K_SetChnGAry (hAD, ChanGainArray.num_of_codes);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global ChanGainArray(16) As Integer
...
' Create the array of channel/gain pairs
ChanGainArray(0) = 2    ' # of chan/gain pairs
ChanGainArray(1) = 0: ChanGainArray(2) = 0
ChanGainArray(3) = 1: ChanGainArray(4) = 1
wDasErr = K_FormatChnGAry (ChanGainArray(0))
wDasErr = K_SetChnGAry (hAD, ChanGainArray(0))
```

K_SetClk

Boards Supported All

Purpose Specifies the pacer clock source.

Prototype

C/C++
DASErr far pascal K_SetClk (DWORD *hFrame*, short *nMode*);

Turbo Pascal for Windows
Function K_SetClk (*hFrame* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetClk Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

nMode Pacer clock source.
Valid values:

Operation	Applicable Boards	Valid Values	Pacer Clock Used
Analog input	All	**0** for Internal	Internal A/D
		1 for External	External
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA boards	**0** for Internal	Internal A/D
	DAS-1700AO Series	**0** for Internal	Internal D/A Internal A/D or burst mode conversion clock[1]
		1 for External	External
Digital I/O	All	**0** for Internal	Internal A/D pacer clock

Notes

[1] If you use K_SetSync to specify that you want to synchronize the analog output operation with the analog input operation, the driver uses the pacer clock used for the analog input operation. The actual clock used depends on the conversion mode.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the pacer clock source in *nMode*.

The *nMode* variable sets the value of the Clock Source element.

The internal A/D pacer clock source is the output of the onboard counter/timer circuitry. The external pacer clock source is an external signal connected to the XPCLK pin (pin 44) on the board's main I/O connector.

If you are performing an analog output operation on a DAS-1700AO Series board and *nMode* = 0, the pacer clock used depends on whether you are synchronizing the analog output operation to the analog input operation. If you are not synchronizing the analog output operation to the analog input operation (the *nClk* parameter of **K_SetSync** = 0), the analog output operation uses the internal D/A pacer clock. If you are synchronizing the analog output operation to the analog input operation (the *nClk* parameter of **K_SetSync** = 1), the pacer clock used depends on the conversion mode specified for the analog input operation (the internal A/D pacer clock for paced mode, the burst mode conversion clock for burst mode or burst mode with SSH).

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame,** and **K_ClearFrame** specify internal as the default clock source. The default active edge is negative for an external clock source; use **K_SetExtClkEdge** to specify a positive active edge.

See Also

K_SetClkRate

For more information about pacer clock sources, refer to the following pages:

Operation	Applicable Boards	Refer to
Analog input	All	page 2-19
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	page 2-37
	DAS-1700AO Series	page 2-50
Digital I/O	All	page 2-61

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetClk (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetClk (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetClk (hAD, 1)
```

Boards Supported

All

Purpose

Specifies the clock rate (number of clock ticks) for the internal pacer clock.

Prototype

C/C++

DASErr far pascal K_SetClkRate (DWORD *hFrame*,
DWORD *dwDivisor*);

Turbo Pascal for Windows

Function K_SetClkRate (*hFrame* : Longint; *dwDivisor* : LongInt) : Word;
far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetClkRate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *dwDivisor* As Long) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

dwDivisor — Number of clock ticks.
Valid values:

Operation	Board Series	Clock Source	Clock Ticks
Analog input	DAS-1700ST Series DAS-1700AO Series	Internal 5 MHz A/D pacer clock	**30** to **4,294,836,225**
	DAS-1700HR Series	Internal 5 MHz A/D pacer clock	**100** to **4,294,836,225**
Analog output	DAS-1700ST-DA Series	Internal 5 MHz A/D pacer clock	**30** to **4,294,836,225**
	DAS-1702HR-DA board	Internal 5 MHz A/D pacer clock	**100** to **4,294,836,225**
	DAS-1700AO Series	Internal 5 MHz D/A pacer clock	**25** to **655,360**
Digital I/O	DAS-1700ST Series DAS-1700AO Series	Internal 5 MHz A/D pacer clock	**30** to **4,294,836,225**
	DAS-1700HR Series	Internal 5 MHz A/D pacer clock	**100** to **4,294,836,225**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the number of clock ticks between conversions in *dwDivisor*.

The *dwDivisor* variable sets the value of the Pacer Clock Rate element.

See Also

K_GetClkRate, K_SetClk

For more information about pacer clock sources, refer to the following pages:

Operation	Applicable Boards	Refer to
Analog input	All	page 2-19
Analog output	DAS-1700ST-DA Series, DAS-1702HR-DA	page 2-37
	DAS-1700AO Series	page 2-50
Digital I/O	All	page 2-61

Usage

C/C++

```
#include "DASDECL.H"
...
DWORD dwClkDiv;
...
dwClkDiv = 5000000 / 10000;
wDasErr = K_SetClkRate (hAD, dwClkDiv);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
dwClkDiv : Longint;
...
dwClkDiv := 5000000 / 10000;
wDasErr := K_SetClkRate (hAD, dwClkDiv);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global dwClkDiv As Long
...
dwClkDiv = 5000000 / 10000
wDasErr = K_SetClkRate (hAD, dwClkDiv)
```

Boards Supported

All

Purpose

Specifies continuous buffering mode.

Prototype

C/C++

DASErr far pascal K_SetContRun (DWORD *hFrame*);

Turbo Pascal for Windows

Function K_SetContRun (*hFrame* : Longint) : Word; far;
external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetContRun Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function sets the buffering mode to continuous mode and sets the Buffering Mode element in the frame accordingly.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame**, and **K_ClearFrame** specify single-cycle as the default buffering mode.

See Also

K_ClrContRun

For more information on buffering modes, refer to the following pages:

Operation	Applicable Boards	Refer to
Analog input	All	page 2-23
Analog output	DAS-1700ST-DA Series DAS-1702HR-DA	page 2-39
	DAS-1700AO	page 2-53
Digital I/O	All	page 2-63

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetContRun (hAD);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetContRun (hAD);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetContRun (hAD)
```

Boards Supported

All

Purpose

Sets up a digital trigger.

Prototype

C/C++

DASErr far pascal K_SetDITrig (DWORD *hFrame*, short *nOpt*, short *nChan*, DWORD *nPattern*);

Turbo Pascal for Windows

Function K_SetDITrig (*hFrame* : Longint; *nOpt* : Word; *nChan* : Word; *nPattern* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetDITrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nOpt* As Integer,
ByVal *nChan* As Integer, ByVal *nPattern* As Long) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nOpt — Trigger polarity and sensitivity.
Valid values:

Operation	Applicable Boards	Valid Values
Analog input	All	**0** for Positive edge **2** for Negative edge
Analog output	DAS-1700AO Series, not using retrigger mode	**0** for Positive edge **2** for Negative edge
	DAS-1700AO Series, using retrigger mode	**4** for Positive edge **6** for Negative edge

nChan — Digital input channel.
Valid value: **0**

nPattern — Trigger pattern.

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the trigger polarity and trigger sensitivity in *nOpt*.

Since the DAS-1700 Series Function Call Driver does not currently support digital pattern triggering, the value of *nPattern* is meaningless. Since the external digital trigger is always connected to the TGIN pin on the DAS-1700 Series boards, the value of *nChan* is meaningless. The *nPattern* and *nChan* parameters are provided for future compatibility.

The *nOpt* variable sets the value of the Trigger Polarity and Trigger Sensitivity elements.

K_SetDITrig does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

If you want to use retrigger mode, specify *nOpt* = 4 or 6. In addition, you must specify recycle mode for the analog output operation by enabling continuous buffering mode and specifying no more than 2048 samples for the operation. Note that you must use **K_ContRun** to specify continuous buffering mode before using *hFrame* as a calling argument to **K_IntStart** or **K_DMAStart**. Refer to page 2-54 for more information about retrigger mode.

See Also

K_SetTrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetDITrig (hAD, 0, 0, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetDITrig (hAD, 0, 0, 0);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetDITrig (hAD, 0, 0, 0)
```

Boards Supported

All

Purpose

Specifies the starting address of a previously allocated buffer and the number of samples in the buffer.

Prototype

C/C++

DASErr far pascal K_SetDMABuf (DWORD *hFrame*, void far **pBuf*, DWORD *dwSamples*);

Turbo Pascal for Windows

Function K_SetDMABuf (*hFrame* : Longint; *pBuf* : Pointer; *dwSamples* : Longint) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetDMABuf Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *pBuf* As Long, ByVal *dwSamples* As Long) As Integer

Parameters

hFrame — Handle to the frame that defines the DMA-mode operation.

pBuf — Starting address of the buffer.

dwSamples — Number of samples.
Valid values:

Operation	Applicable Boards	Operation Mode	# of Samples
Analog input	All	DMA	1 to 65536
Analog output	DAS-1700AO Series	DMA	1 to 65536
		Recycle	1 to 2048

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation specified by *hFrame*, this function specifies the starting address of a previously allocated buffer in *pBuf* and the number of samples in the buffer in *dwSamples*.

The *pBuf* variable sets the value of the Buffer element; the *dwSamples* variable sets the value of the Number of Samples element.

See Also

K_DMAAlloc, KMakeDMABuf, K_BufListAdd

Usage

C/C++

```
#include "DASDECL.H"
...
void far *pBuf;    // Pointer to allocated buffer
...
wDasErr = K_DMAAlloc (hAD, 1000, &pBuf, &hMem);
wDasErr = K_SetDMABuf (hAD, pBuf, 1000);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
TYPE
BufType = Array [0..1] of Integer;
VAR
pBuf : ^BufType;    { buffer pointer }
...
wDasErr := K_DMAAlloc (hAD, 1000, Addr(pBuf), hMem);
wDasErr := K_SetDMABuf (hAD, pBuf, 1000);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
Global pBuf As Long
...
wDasErr = K_DMAAlloc (hAD, 1000, pBuf, hMem)
wDasErr = K_SetDMABuf (hAD, pBuf, 1000)
```

Boards Supported

All

Purpose

Specifies the active edge of the external pacer clock.

Prototype

C/C++

DASErr far pascal K_SetExtClkEdge (DWORD *hFrame*, short *nEdge*);

Turbo Pascal for Windows

Function K_SetExtClkEdge (*hFrame* : Longint; *nEdge* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetExtClkEdge Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nEdge* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nEdge	Active edge of external pacer clock. Valid values: **0** = Negative edge **1** = Positive edge

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the active edge of the external pacer clock in *nEdge,* and sets the External Clock Edge element in the frame accordingly.

K_SetExtClkEdge does not affect the operation defined by *hFrame* unless the Clock Source element is set to External (by a call to **K_SetClk**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**.

See Also

K_SetClk

K_SetExtClkEdge (cont.)

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetExtClkEdge (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetExtClkEdge (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetExtClkEdge (hAD, 1)
```

Boards Supported All

Purpose Sets the gain.

Prototype

C/C++
DASErr far pascal K_SetG (DWORD *hFrame*, short *nGain*);

Turbo Pascal for Windows
Function K_SetG (*hFrame* : Longint; *nGain* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetG Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nGain* As Integer) As Integer

Parameters

hFrame Handle to the frame that defines the operation.

nGain Gain code.
Valid values:

Operation	Applicable Boards	Valid Values	For a table of ranges and gain codes, refer to
Analog input	All	**0** to **3** for DAS board channels **0** to **7** for EXP-1800 channels	Table 2-2 on page 2-13
Analog output	DAS-1700AO Series	**0** or **1**	Table 2-5 on page 2-47

Return Value Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the gain code for a single channel or for a group of consecutive channels in *nGain*.

The *nGain* variable sets the value of the Gain element.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame**, and **K_ClearFrame K_ClearFrame** specify a gain of 1 (gain code 0) as the default gain.

For the DAS-1700ST/HR Series boards, this function is valid for A/D frames only. For the DAS-1700AO Series boards, this function is valid for A/D and D/A frames.

See Also

K_SetStartStopG

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetG (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetG (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetG (hAD, 1)
```

Boards Supported

All

Purpose

Specifies the status of the hardware gate.

Prototype

C/C++

DASErr far pascal K_SetGate (DWORD *hFrame*, short *nMode*);

Turbo Pascal for Windows

Function K_SetGate (*hFrame* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetGate Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nMode	Status of the hardware gate. Valid values: **0** = Gate disabled **1** = Positive gate enabled **2** = Negative gate enabled

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the status of the hardware gate in *nMode*.

All DAS-1700 Series boards support external gating for analog input operations. The DAS-1700AO Series boards also support external gating for analog output operations.

You cannot enable the hardware gate if you are using an external digital trigger.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame**, and **K_ClearFrame** specify disabled as the default gate setting.

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetGate (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetGate (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetGate (hAD, 1)
```

Boards Supported

All

Purpose

Enables and disables burst mode with SSH.

Prototype

C/C++

DASErr far pascal K_SetSSH (DWORD *hFrame*, WORD *nMode*);

Turbo Pascal for Windows

Function K_SetSSH (*hFrame* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetSSH Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nMode	Status of burst mode with SSH. Valid values: **0** = Disabled **1** = Enabled

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the status of burst mode with SSH in *nMode*.

K_GetADFrame and **K_ClearFrame** disable burst mode with SSH.

Refer to page 2-18 for information on conversion modes.

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetSSH (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetSSH (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetSSH (hAD, 1)
```

Boards Supported

All

Purpose

Specifies the first and last channels in a group of consecutive channels.

Prototype

C/C++

DASErr far pascal K_SetStartStopChn (DWORD *hFrame*, short *nStart*, short *nStop*);

Turbo Pascal for Windows

Function K_SetStartStopChn (*hFrame* : Longint; *nStart* : Word; *nStop* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetStartStopChn Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nStart* As Integer, ByVal *nStop* As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nStart — First channel in a group of consecutive channels. Valid values:

Operation	Applicable Boards	Valid Values
Analog input	All, with no EXP-1800 expansion accessories attached	Differential: **0** to **7** Single-ended: **0** to **15**
	All, with *n* EXP-1800 expansion accessories attached	**0** to **15**(***n*** + **1**)
Analog output	DAS-1700ST-DA Series	**0** to **3**
	DAS-1700AO Series DAS-1702HR-DA boards	**0** to **1**

nStop — Last channel in a group of consecutive channels. Valid values: Same as for *nStart* above

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the first channel in a group of consecutive channels in *nStart* and the last channel in the group of consecutive channels in *nStop*.

For analog input operations, the start channel can be higher than the stop channel.

For analog output operations on the DAS-1700ST-DA Series and the DAS-1702HR-DA boards, the start channel must be lower than or equal to the stop channel.

For analog output operations on the DAS-1700AO Series boards, the start channel must be channel 0 (DAC 0) and the stop channel must be channel 1 (DAC1).

The *nStart* variable sets the value of the Start Channel element; the *nStop* variable sets the value of the Stop Channel element.

K_GetADFrame, **K_GetDAFrame**, **K_GetDIFrame**, **K_GetDOFrame** and **K_ClearFrame** set the Start Channel and Stop Channel elements to 0.

See Also

K_SetChn, K_SetStartStopG

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetStartStopChn (hAD, 0, 7);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetStartStopChn (hAD, 0, 7);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetStartStopChn (hAD, 0, 7)
```

Boards Supported

All

Purpose

Specifies the first and last channels in a group of consecutive channels and sets the gain for all channels in the group.

Prototype

C/C++

DASErr far pascal K_SetStartStopG (DWORD *hFrame*, short *nStart*, short *nStop*, short *nGain*);

Turbo Pascal for Windows

Function K_SetStartStopG (*hFrame* : Longint; *nStart* : Word; *nStop* : Word; *nGain* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetStartStopG Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nStart* As Integer, ByVal *nStop* As Integer, ByVal *nGain* As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nStart — First channel in a group of consecutive channels. Valid values:

Operation	Applicable Boards	Valid Values
Analog input	All, with no EXP-1800 expansion accessories attached	Differential: **0** to **7** Single-ended: **0** to **15**
	All, with *n* EXP-1800 expansion accessories attached	**0** to **15**(***n*** + **1**)
Analog output	DAS-1700ST-DA Series	**0** to **3**
	DAS-1700AO Series DAS-1702HR-DA boards	**0** to **1**

nStop — Last channel in a group of consecutive channels. Valid values: Same as for *nStart* above

nGain Gain code.
Valid values:

Operation	Applicable Boards	Valid Values	For a table of ranges and gain codes, refer to
Analog input	All	**0** to **3** for DAS board channels **0** to **7** for EXP-1800 channels	Table 2-2 on page 2-13
Analog output	DAS-1700AO Series	**0** or **1**	Table 2-5 on page 2-47

Return Value Error/status code. Refer to Appendix A.

Remarks For the operation defined by *hFrame*, this function specifies the first channel in a group of consecutive channels in *nStart*, the last channel in a group of consecutive channels in *nStop*, and the gain code for all channels in the group in *nGain*.

For analog input operations, the start channel can be higher than the stop channel.

For analog output operations on the DAS-1700ST-DA Series and the DAS-1702HR-DA boards, the start channel must be lower than or equal to the stop channel.

For analog output operations on the DAS-1700AO Series boards, the start channel must be channel 0 (DAC 0) and the stop channel must be channel 1 (DAC1).

The *nStart* variable sets the value of the Start Channel element; the *nStop* variable sets the value of the Stop Channel element; the *nGain* variable sets the value of the Gain element.

K_GetADFrame and **K_ClearFrame** set the Start Channel, Stop Channel, and Gain elements to zero.

See Also K_SetChn, K_SetStartStopChn

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetStartStopG (hAD, 0, 7, 0);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetStartStopG (hAD, 0, 7, 0);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetStartStopG (hAD, 0, 7, 0)
```

K_SetSync

Boards Supported

DAS-1701AO, DAS-1702AO

Purpose

Specifies the operation to which an analog output operation is synchronized.

Prototype

C/C++

DASErr far pascal K_SetSync (DWORD *hFrame*, short *nClk*);

Turbo Pascal for Windows

Function K_SetSync (*hFrame* : Longint; *nClk* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetSync Lib "DASSHELL.DLL" (ByVal *hFrame* As Long, ByVal *nClk* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the analog output operation.
nClk	Operation to which the analog output operation is synchronized. Valid values: **0** for None **1** for Analog input operation

Return Value

Error/status code. Refer to Appendix A.

Remarks

In *nClk*, this function specifies the operation to which the analog output operation defined by *hFrame* is synchronized.

If *nClk* = 1, you must specify an internal clock for both the analog input operation and the analog output operation (the *nMode* parameter of **K_SetClk** must = 0 for both). The clock used to determine the update rate for the synchronized analog output operation depends on the conversion mode specified for the analog input operation. If you specified paced mode, the internal A/D pacer clock is used; if you specified burst mode or burst mode with SSH, the burst mode conversion clock is used. Refer to page 2-52 for more information.

Usage

C/C++

```
#include "DASDECL.H"    // Use "DASDECL.HPP for C++
...
wDasErr = K_SetSync (hDA, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetSync (hDA, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS to your project)

```
...
wDasErr = K_SetSync (hDA, 1)
```

K_SetTrig

Boards Supported

All

Purpose

Specifies the start trigger source.

Prototype

C/C++
DASErr far pascal K_SetTrig (DWORD *hFrame*, short *nMode*);

Turbo Pascal for Windows
Function K_SetTrig (*hFrame* : Longint; *nMode* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows
Declare Function K_SetTrig Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nMode* As Integer) As Integer

Parameters

hFrame	Handle to the frame that defines the operation.
nMode	Trigger source. Valid values: **0** = Internal trigger **1** = External trigger

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the trigger source in *nMode*.

An internal trigger is a software trigger; the trigger event occurs when the operation is started. An external trigger is either an analog trigger or a digital trigger. Refer to page 2-24 for more information about trigger sources for analog input operations on all DAS-1700 Series boards or to page 2-54 for more information about trigger sources for analog output operations (DAS-1700AO Series boards only).

If *nMode* = **1**, an external digital trigger (positive edge on TGIN) is assumed. Use **K_SetDITrig** to change the active edge of the digital trigger. Use **K_SetADTrig** to specify the conditions for an external analog trigger.

K_GetADFrame, K_GetDAFrame (DAS-1700AO Series only), and **K_ClearFrame** set the trigger source to internal.

For all DAS-1700 Series boards, the external trigger source is relevant for analog input operations. For the DAS-1700AO Series boards, the external trigger source is also relevant to analog output operations.

See Also

K_SetADTrig, K_SetDITrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetTrig (hAD, 1);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetTrig (hAD, 1);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetTrig (hAD, 1)
```

K_SetTrigHyst

Boards Supported

All

Purpose

Specifies the hysteresis value.

Prototype

C/C++

DASErr far pascal K_SetTrigHyst (DWORD *hFrame*, short *nHyst*);

Turbo Pascal for Windows

Function K_SetTrigHyst (*hFrame* : Longint; *nHyst* : Word) : Word; far; external 'DASSHELL';

Visual Basic for Windows

Declare Function K_SetTrigHyst Lib "DASSHELL.DLL"
(ByVal *hFrame* As Long, ByVal *nHyst* As Integer) As Integer

Parameters

hFrame — Handle to the frame that defines the operation.

nHyst — Hysteresis value, specified in counts.
Valid values:

DAS-1700ST Series DAS-1700AO Series	**0** to **4095**
DAS-1700HR Series	**0** to **65535**

Return Value

Error/status code. Refer to Appendix A.

Remarks

For the operation defined by *hFrame*, this function specifies the hysteresis value used for an analog trigger in *nHyst*.

You specify the hysteresis value in counts. Refer to Appendix B for information on converting the hysteresis voltage to a count value.

For DAS-1700HR Series boards, the count value for *nHyst* can be as high as 65,535. Since Visual Basic for Windows treats integers as signed numbers, you must represent values greater than 32,767 as negative integers. For information on converting these values, refer to page 3-24.

The *nHyst* variable sets the value of the Trigger Hysteresis element.

K_SetTrigHyst does not affect the operation defined by *hFrame* unless the Trigger Source element is set to External (by a call to **K_SetTrig**) before *hFrame* is used as a calling argument to **K_IntStart** or **K_DMAStart**. Refer to page 2-25 for more information about analog triggers.

See Also

K_SetADTrig

Usage

C/C++

```
#include "DASDECL.H"
...
wDasErr = K_SetTrigHyst (hAD, 50);
```

Turbo Pascal for Windows

```
{$I DASDECL.INC}
...
wDasErr := K_SetTrigHyst (hAD, 50);
```

Visual Basic for Windows

(Add DASDECL.BAS or DASDEC32.BAS to your project)

```
...
wDasErr = K_SetTrigHyst (hAD, 50)
```

A

Error/Status Codes

Table A-1 lists the error/status codes that are returned by the DAS-1700 Series Function Call Driver, possible causes for error conditions, and possible solutions for resolving error conditions.

If you cannot resolve an error condition, contact Keithley MetraByte for technical support.

Table A-1. Error/Status Codes

Error Code		Cause	Solution
Hex	Decimal		
0	0	No error has been detected.	Status only; no action is necessary.
6000	24576	**Error in configuration file:** The configuration file you specified in the driver initialization function is corrupt, does not exist, or contains one or more undefined keywords.	Check that the file exists at the specified path. Check for illegal keywords in file; you can avoid illegal keywords by using the configuration utility to create and modify configuration files.
6001	24577	**Illegal base address in configuration file:** The board's base I/O address in the configuration file is illegal and/or does not match the base address switches on the board.	Use the configuration utility to change the base I/O address to one that matches the base address switches on the board.
6002	24578	**Illegal IRQ level in configuration file:** The interrupt level in the configuration file is illegal.	Use the configuration utility to change the interrupt level to a legal one for your board. Refer to the user's guide for legal interrupt levels.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6003	24579	**Illegal DMA channel in configuration file:** The DMA channel in the configuration file is illegal.	Use the configuration utility to change the DMA channel to a legal one for your board. Refer to the user's guide for legal DMA channels.
6005	24581	**Illegal channel number:** The specified channel number is illegal for the board and/or for the range type (unipolar or bipolar).	Specify a legal channel number. Refer to the user's guide or to the description of **K_SetStartStopChn** in Chapter 4 for legal channel numbers.
6006	24582	**Illegal gain code:** The specified analog I/O channel gain code is illegal for this board.	Specify a legal gain code. Refer to the user's guide or to the description of **K_SetG** in Chapter 4 for a list of legal gain codes.
6007	24583	**Illegal DMA address:** An FCD function specified a buffer address that is not suitable for a DMA operation for the number of samples required.	Use the **K_DMAAlloc** function to allocate dynamic buffers for DMA operations. In Windows, make sure that the Keithley Memory Manager is installed; refer to Appendix D of the user's guide for information.
6008	24584	**Illegal number in configuration file:** The configuration file contains one or more numeric values that are illegal.	Use the configuration utility to check and then change the configuration file.
600A	24586	**Configuration file not found:** The driver cannot find the configuration file specified as an argument to the driver initialization function.	Check that the file exists at the specified path. Check that the file name is spelled correctly in the driver initialization function parameter list.
600B	24587	**Error returning DMA buffer:** DOS returned an error in INT 21H function 49H during the execution of **K_DMAFree**.	Check that the memory handle passed as an argument to **K_DMAFree** was previously obtained using **K_DMAAlloc**.
600C	24588	**Error returning interrupt buffer:** The memory handle specified in **K_IntFree** is invalid.	Check the memory handle stored by **K_IntAlloc** and make sure that it was not modified.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
600D	24589	**Illegal frame handle:** The specified frame handle is not valid for this operation.	Check that the frame handle exists. Check that you are using the appropriate frame handle.
600E	24590	**No more frame handles:** No frames are left in the pool of available frames.	The Function Call Driver supports a maximum of 30 frames of all types. Use **K_FreeFrame** to free a frame that the application is no longer using.
600F	24591	**Requested buffer size too large:** The requested buffer cannot be dynamically allocated because of its size.	Specify a smaller buffer size; refer to the description of **K_IntAlloc** in Chapter 4 for the legal range. If the Keithley Memory Manager is installed, use KMMSETUP.EXE to increase the reserved buffer heap size.
6010	24592	**Cannot allocate interrupt buffer:** (Windows-based languages only) **K_IntAlloc** failed because there was not enough available DOS memory.	Remove some Terminate and Stay Resident programs (TSRs) that are no longer needed.
6012	24594	**Interrupt buffer deallocation error:** (Windows-based languages only) An error occurred when **K_IntFree** attempted to free a memory handle.	Make sure that the memory handle passed as an argument to **K_IntFree** was previously obtained using **K_IntAlloc**.
6015	24597	**DMA Buffer too large:** The number of samples specified in **K_DMAAlloc** is too large.	Refer to the description of **K_DMAAlloc** in Chapter 4 for the buffer size range.
6016	24598	**VDS - Region not contiguous:** An error occurred while using Windows Virtual DMA Services. You tried to use **K_DMAAlloc** and the Keithley Memory Manager was not installed.	Refer to Appendix D in the user's guide for information on how to install and set up the Keithley Memory Manager.
6017	24599	**VDS - DMA wraparound:** See error 6016.	See error 6016.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6018	24600	**VDS - Unable to lock region:** See error 6016.	See error 6016.
6019	24601	**VDS - No buffer available:** See error 6016.	See error 6016.
601A	24602	**VDS - Region too large:** See error 6016.	See error 6016.
601B	24603	**VDS - Buffer in use:** See error 6016.	See error 6016.
601C	24604	**VDS - Illegal region:** See error 6016.	See error 6016.
601D	24605	**VDS - Region not locked:** See error 6016.	See error 6016.
601E	24606	**VDS - Illegal page:** See error 6016.	See error 6016.
601F	24607	**VDS - Illegal buffer:** See error 6016.	See error 6016.
6020	24608	**VDS - Copy out of range:** See error 6016.	See error 6016.
6021	24609	**VDS - Illegal DMA channel:** See error 6016.	See error 6016.
6022	24610	**VDS - Count overflow:** See error 6016.	See error 6016.
6023	24611	**VDS - Count underflow:** See error 6016.	See error 6016.
6024	24612	**VDS - Function not supported:** See error 6016.	See error 6016.
6025	24613	**Illegal OBM mode:** The mode number specified in **K_SetOBMMode** is illegal.	Specify a legal mode value.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6026	24614	**Illegal DMA structure:** An error occurred during the execution of **K_DMAFree**.	Try using **K_DMAFree** again. If the error continues, contact Keithley MetraByte for technical support.
6027	24615	**DMA allocation error:** See error 6026.	See error 6026.
6028	24616	**NULL DMA handle:** See error 6026.	See error 6026.
6029	24617	**DMA unlock error:** See error 6026.	See error 6026.
602A	24618	**DMA free error:** See error 6026.	See error 6026.
602B	24619	**Not enough memory to accommodate request:** The number of samples you requested is greater than the largest contiguous block available in the reserved heap.	Specify a smaller number of samples. Free a previously allocated buffer. Use the KMMSETUP.EXE utility to expand the reserved heap.
602C	24620	**Requested buffer size exceeds maximum:** The number of samples you requested is greater than the allowed maximum.	Specify a value within the legal range when calling **K_DMAAlloc**. Refer to the description of **K_DMAAlloc** in Chapter 4 for legal values.
602D	24621	**Illegal device handle:** A bad device handle was passed to a function such as **K_GetADFrame**. The handle used was not initialized through a call to a card/board initialization function, such as **K_GetDevHandle**, or it was corrupted by your program.	Check the device handle value.
602E	24622	**Illegal setup option:** An illegal option was specified to a function that accepts a user option, such as **K_SetDITrig**.	Check the option value passed to the function where the error occurred.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6030	24624	**DMA word-page wrap:** During **K_DMAAlloc**, a DMA word-page wrap condition occurred and the allocation attempt failed since there is not enough free memory to accommodate the allocation request.	Reduce the number of samples and retry. Install and configure the Keithley Memory Manager.
6031	24625	**Illegal memory handle:** A bad memory handle was passed to **K_IntFree** or **K_DMAFree**. The handle used was not initialized through a call to **K_IntAlloc** or **K_DMAAlloc**, or it was corrupted by your program.	Restart your program and monitor the memory handle value(s).
6032	24626	**Out of memory handles:** An attempt to allocate a memory block using **K_IntAlloc** or **K_DMAAlloc** failed because the maximum number of handles has already been assigned.	Use **K_IntFree** or **K_DMAFree** to free previously allocated memory blocks before allocating memory again.
6034	24628	**Memory corrupted:** Int 21H function 48H, used to allocate a memory block from the DOS far heap, returned the DOS error 7; this means that memory is corrupted. It is likely that you stored data (through a DMA-mode or interrupt-mode operation) into an illegal area of DOS memory.	Recheck the parameters set by **K_DMAAlloc** and **K_SetDMABuf**, or by **K_IntAlloc** and **K_SetBuf**. If a fatal system error occurred, restart your computer.
6035	24629	**Driver in use:** You attempted to initialize a driver that was already initialized by a call to **K_OpenDriver**. (This can occur since, under Windows, it is possible to open the same driver from multiple programs that are running simultaneously.)	To continue using the driver with the same configuration, pass a null string as the second argument to **K_OpenDriver**. To use the driver with a different configuration, close any application programs currently accessing the driver, and then open the driver again (using **K_OpenDriver**).

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
6036	24630	**Illegal driver handle:** The specified driver handle is not valid.	Someone may have closed the driver; if so, use **K_OpenDriver** to reopen the driver with the desired driver handle. Try again using another driver handle.
6037	24631	**Driver not found:** The specified driver cannot be found.	Check your link statement to make sure the specified driver is included. Make sure that the device name string is entered correctly in **K_OpenDriver**.
6038	24632	**Invalid source pointer:** (Windows-based languages only) The pointer to the source buffer that you passed as an argument to **K_MoveBufToArray** is invalid for the specified count. (The source pointer, when added to the number of samples, exceeds the programmed addressing range of that pointer.)	Check the pointer to the source buffer and the number of samples to transfer that you specified in **K_MoveBufToArray**.
6039	24633	**Invalid destination pointer:** (Windows-based languages only) The pointer to the destination buffer (local array) that you passed as an argument to **K_MoveBufToArray** is invalid for the specified count. (The destination pointer, when added to the number of samples, exceeds the dimension of the local array.)	Check the dimension of the local array and the number of samples to transfer that you specified in **K_MoveBufToArray**.
603A	24634	**Illegal setup value:** An illegal value was passed to the function in which the error occurred.	Check the legal ranges of all parameters passed to this function.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
603B	24635	**Error freeing buffer selector: K_DMAFree** or **K_IntFree** failed because one or more of the selectors that reference the memory buffer could not be freed.	Check that the memory buffer being freed was previously allocated using **K_DMAAlloc** or **K_IntAlloc.**
603C	24636	**Error allocating buffer selector: K_DMAAlloc** or **K_IntAlloc** failed because a selector could not be allocated from Window's Local Descriptor Table.	Close all applications and restart Windows. If the error continues, contact Keithley MetraByte for technical support.
603D	24637	**Error allocating memory buffer: K_DMAAlloc** or **K_IntAlloc** failed because a necessary internal buffer could not be allocated to complete the operation. You attempted to specify the starting address of a locally dimensioned array in a 32-bit Windows 95 program.	Close all programs and restart Windows. In a Windows 95, 32-bit program, make sure that you use **K_IntAlloc** or **K_DMAAlloc** to dynamically allocate a memory buffer and make sure that you use **K_SetBuf** or **K_BufListAdd** to specify the starting address of the dynamically allocated memory buffer. If the error continues, contact Keithley MetraByte for technical support.
603Fh	24639	**Out of device handles:** You attempted to initialize a board with **K_GetDevHandle,** but no device handles were available for the driver to return.	Use **K_FreeDevHandle** to free a device handle that is no longer in use.
6040h	24640	**Out of driver handles:** You attempted to initialize a driver with **K_OpenDriver**, but no driver handles were available for the driver to return.	Use **K_CloseDriver** to free a driver handle that is no longer being used.
7000	28672	**No board name**: The driver initialization function did not find a board name in the specified configuration file.	Specify a legal board name in the configuration file.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7001	28673	**Illegal board name**: The board name in the specified configuration file is illegal.	Specify a legal board name in the configuration file.
7002	28674	**Illegal board number**: The driver initialization function found an illegal board number in the specified configuration file.	Specify a legal board number: 0, 1, or 2.
7003	28675	**Illegal base address**: The driver initialization function found an illegal base address in the specified configuration file.	Specify a base address in the inclusive range &H200 (512) to &H3F0 (1008) in increments of 10h (16). Make sure that &H precedes hexadecimal numbers.
7004	28676	**Illegal DMA channel**: The driver initialization function found an illegal DMA channel in the specified configuration file.	Specify a legal DMA channel: 5, 6, 7, 5+6, 6+7, or 7+5.
7005	28677	**Illegal interrupt level**: The driver initialization function found an illegal interrupt level in the specified configuration file.	Specify a legal interrupt level: 3, 5, 7, 10, 11, or 15.
7006	28678	**Illegal digital trigger and gate combination**: You attempted to use a hardware gate and an external digital trigger in the same operation.	Either disable the hardware gate using the **K_SetGate** function, or specify a different type of trigger. To specify an internal trigger, use the **K_SetTrig** function to set the trigger source to internal. To specify an external analog trigger, use the **K_SetTrig** function to set the source to external and then use the **K_SetADTrig** function to specify the conditions for the external analog trigger.
7007	28679	**Illegal A/D channel mode**: The driver initialization function found an illegal input range type in the specified configuration file.	Specify a legal input range type: bipolar, unipolar.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7008	28680	**Illegal A/D channel configuration**: The driver initialization function found an illegal input configuration in the specified configuration file.	Specify a legal input configuration: single-ended, differential.
7009	28681	**Illegal channel number**: The specified channel is out of range.	Specify a legal channel number. Refer to the user's guide or to the description of **K_SetStartStopChn** in Chapter 4 for legal channel numbers.
700A	28682	**Illegal number of SSH8**: The number of SSH-8s in the configuration file is not valid.	Specify a legal number of SSH-8s: 0 to 8.
700B	28683	**Illegal SSH-8 channel**: The SSH-8 channel in the configuration file is not valid.	Specify a legal SSH-8 channel number: 0 to 7.
700C	28684	**Illegal SSH-8 gain**: The SSH-8 channel gain in the configuration file is not valid.	Specify a legal SSH-8 channel gain: 0.5, 5, 50 or 250.
700E	28686	**Resource busy**: The application program attempted to start an operation while a similar operation was in progress.	Use **K_IntStop** or **K_DMAStop** to stop the in-progress operation before initiating the second operation.
700F	28687	**Illegal analog trigger, A/D busy:** An analog input operation was in progress when the application attempted to start an analog output operation for which an analog trigger was defined.	Wait until the analog input operation is done, or use **K_IntStop** or **K_DMAStop** to stop the analog input operation before initiating the analog output operation.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7010	28688	**Illegal re-trigger mode:** You enabled retrigger mode in **K_SetDITrig**, but either the number of output values stored in the user-defined buffer exceeds 2048 or single-cycle buffering mode is specified.	If you want to enable retrigger mode, make sure that the user-defined buffer contains 2048 or fewer output values and make sure that continuous buffering mode is enabled. If you do not want to enable retrigger mode, make sure that you specify an appropriate trigger polarity and sensitivity in **K_SetDITrig**.
7011	28689	**D/A FIFO Underflow:** The pacer clock rate specified for an analog output operation is too fast.	Specify a slower pacer clock rate.
7012	28690	**Illegal burst mode conversion clock divider**: The burst rate divider passed to **K_SetBurstTicks** is out of range.	Specify a burst rate divider in the range of 3 to 64.
7013	28691	**DMA channel busy**: The application program attempted to start a DMA-mode analog input operation while another DMA-mode analog input operation was active.	Use **K_DMAStop** to stop the active operation before initiating the second operation.
7014	28692	**Counter 0 resource busy**: The application program attempted to start a DMA-mode analog input operation with about-trigger mode enabled while another DMA-mode operation with about-trigger was active.	Use **K_DMAStop** to stop the active operation before initiating the second operation.
7015	28693	**Illegal number of about-trigger samples**: The number of samples passed to **K_SetAboutTrig** is out of range.	Specify a number of samples in the range of 1 to 65,535.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
7016	28694	**Illegal about-trigger mode**: About-trigger mode was enabled for an interrupt-mode operation.	Use K_ClrAboutTrig to disable about-trigger mode.) Perform this about-trigger acquisition in DMA mode.
7017	28695	**Illegal number of EXP-1800 boards**: The number of EXP-1800 expansion accessories specified in the configuration file is not valid.	Specify a legal number of EXP-1800 expansion accessories: 0 to 16.
8001	32769	**Function not supported:** You attempted to use a function not supported by the Function Call Driver.	Contact Keithley MetraByte for technical support.
8003	32771	**Illegal board number:** An illegal board number was specified in a board initialization function (such as **K_GetDevHandle**).	Refer to the description of **K_GetDevHandle** in Chapter 4 for legal board numbers.
8004	32772	**Illegal error number:** The error message number specified in **K_GetErrMsg** is invalid.	The error number must be one the error numbers listed in this appendix.
8005	32773	**Board not found at configured address:** The board initialization function does not detect the presence of a board.	Make sure that the base address setting of the switches on the board matches the base address setting in the configuration file.
8006	32774	**A/D not initialized:** You attempted to start a frame-based analog input operation without the A/D frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
8007	32775	**D/A not initialized:** You attempted to start a frame-based analog output operation without the D/A frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
8008	32776	**Digital input not initialized:** You attempted to start a frame-based digital input operation without the DI frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
8009	32777	**Digital output not initialized:** You attempted to start a frame-based digital output operation without the DO frame being properly initialized.	Always call **K_ClearFrame** before setting up a new frame-based operation.
800B	32779	**Conversion overrun:** Data was overwritten before it was transferred to the computer's memory.	Adjust the clock source to slow down the rate at which the board acquires data. Remove other programs that are running and using computer resources. If running in interrupt mode, check that the number of samples is a multiple of 512.
8016	32790	**Interrupt overrun**: The board communicated a hardware event to the software by generating a hardware interrupt, but the software was still servicing a previous interrupt. This is usually caused by a pacer clock rate that is too fast.	Check the maximum throughput rate for your computer's programming environment and use **K_SetClkRate** to specify an appropriate rate.
801A	32794	**Interrupts already active:** You attempted to start an operation whose interrupt level is being used by another system resource.	Use **K_IntStop** to stop the first operation before starting the second operation.
801B	32795	**DMA already active**: You attempted to start a DMA-mode operation using a DMA channel that is currently used by another active operation.	Use **K_DMAStop** to stop the first operation before starting the second operation.
8020	32800	**FIFO Overflow event detected:** During data acquisition, the temporary on-board data storage (FIFO) overflowed.	The conversion rate is too fast for your computer's programming environment; use **K_SetClkRate** to reduce the conversion rate. If you are using DMA-mode and your board supports dual-DMA, use the configuration utility to reconfigure your board to use dual-DMA.

Table A-1. Error/Status Codes (cont.)

Error Code		Cause	Solution
Hex	Decimal		
8021	32801	**Illegal clock sync mode**: The two operations you are trying to synchronize cannot be synchronized on your board.	Check the synchronizing operation that you specified in **K_SetSync**. Make sure that your board supports the synchronization of the two operations.
FFFF	65535	**User aborted operation:** You pressed **Ctrl+Break** during a synchronous-mode operation or while waiting for an analog trigger event to occur.	Start the operation again, if desired.

B

Data Formats

This appendix contains the following sections:

- **Converting Counts to Voltage** - instructions for converting a count value returned by the DAS-1700 Series Function Call Driver to a voltage value.
- **Converting Voltage to Counts** - instructions for converting a voltage value to a count value that the DAS-1700 Series Function Call Driver can understand.

Converting Counts to Voltage

You may want to convert counts to voltage when reading an analog input value.

To convert an analog input value to a voltage, use one of the following equations, where *count* is the count value, and *span* is the span in volts (refer to Table B-1 on page B-2):

DAS-1700ST Series and DAS-1700AO Series Boards

$$\text{Voltage} = \frac{\text{count} \times \text{span}}{4096}$$

DAS-1700HR Series Boards

$$\text{Voltage} = \frac{\text{count} \times \text{span}}{65536}$$

Table B-1. Span Values For Data Conversion Equations

Board	Input Range Type	Gain	Input Range	Span (V)
DAS-1701ST DAS-1701ST-DA DAS-1701AO	Unipolar	1	0 to 5 V	5
		5	0 to 1 V	1
		50	0 to 0.1 V	0.1
		250	0 to 0.02 V	0.02
	Bipolar	1	–5 to 5 V	10
		5	–1 to 1 V	2
		50	–0.1 to 0.1 V	0.2
		250	–0.02 to 0.02 V	0.04
DAS-1702ST DAS-1702ST-DA DAS-1702HR DAS-1702HR-DA DAS-1702AO	Unipolar	1	0 to 10 V	10
		2	0 to 5 V	5
		4	0 to 2.5 V	2.5
		8	0 to 1.25 V	1.25
	Bipolar	1	–10 to 10 V	20
		2	–5 to 5 V	10
		4	–2.5 to 2.5 V	5
		8	–1.25 to 1.25 V	2.5
DAS-1701ST DAS-1701ST-DA DAS-1701AO (attached to an EXP-1800 using a gain of 50)	Unipolar	50	0 to 0.1 V	0.1
		250	0 to 0.02 V	0.02
		2500	0 to 0.002 V	0.002
		12.5k	0 to 0.0004 V	0.0004
	Bipolar	50	–0.1 V to 0.1 V	0.2
		250	–0.02 V to 0.02 V	0.04
		2500	–0.002 V to 0.002 V	0.004
		12.5k	–0.0004 V to 0.0004 V	0.0008

Table B-1. Span Values For Data Conversion Equations (cont.)

Board	Input Range Type	Gain	Input Range	Span (V)
DAS-1702ST DAS-1702ST-DA DAS-1702HR DAS-1702HR-DA DAS-1702AO (attached to an EXP-1800 using a gain of 50)	Unipolar	50	0 to 0.2 V	0.2
		100	0 to 0.1 V	0.1
		200	0 to 50 mV	0.05
		400	0 to 25 mV	0.025
	Bipolar	50	–0.2 V to 0.2 V	0.4
		100	–0.1 V to 0.1 V	0.2
		200	–50 mV to 50 mV	0.1
		400	–25 mV to 25 mV	0.05

For example, assume that you want to read analog input data from a channel on a DAS-1701ST board configured for a unipolar input range type; the channel collects the data at a gain of 1. The count value is 3072. The voltage is determined as follows:

$$\frac{3072 \times 5 \text{ V}}{4096} = 3.75 \text{ V}$$

As another example, assume that you want to read analog input data from a channel on a DAS-1702ST board configured for a bipolar input range type; the channel collects the data at a gain of 2. The count value is 1024. The voltage is determined as follows:

$$\frac{1024 \times 10 \text{ V}}{4096} = 2.5 \text{ V}$$

Converting Voltage to Counts

You must convert voltage to counts when specifying an analog output value, analog trigger level, or hysteresis value.

Specifying an Analog Output Value

To convert a voltage value to a count when specifying an analog output value, use the following equation, where *voltage* is the desired voltage:

DAS-1700ST-DA Series and DAS-1700AO Series Boards

$$\text{Count} = \frac{\text{voltage} \times 4096}{20\ \text{V}}$$

DAS-1702HR-DA Board

$$\text{Count} = \frac{\text{voltage} \times 65536}{20\ \text{V}}$$

To specify an analog output value of 5 V for a channel on a DAS-1702ST-DA, enter a count of 1024, determined as follows:

$$\frac{5\ \text{V} \times 4096}{20\ \text{V}} = 1024$$

To specify an analog output value of 5 V for a channel on a DAS-1702HR-DA, enter a count of 16384, determined as follows:

$$\frac{5\ \text{V} \times 65536}{20\ \text{V}} = 16384$$

Specifying an Analog Trigger Level

To convert a voltage value to a count when specifying an analog trigger level, use one of the following equations, where V_{trig} is the desired voltage, and *span* is the appropriate value from Table B-1 on page B-2:

DAS-1700ST Series and DAS-1700AO Series Boards

$$\text{Count} = \frac{V_{trig} \times 4096}{\text{span}}$$

DAS-1700HR Series Boards

$$\text{Count} = \frac{V_{trig} \times 65536}{\text{span}}$$

Note: When converting voltage to counts to specify an analog trigger level, always use a gain of 1 to determine which span value to use from Table B-1, no matter what the gain of the channel.

For example, assume that you want to specify an analog trigger level of 2.5 V for a channel on a DAS-1701ST board configured for a bipolar input range type. The count is determined as follows:

$$\frac{2.5\ \text{V} \times 4096}{10\ \text{V}} = 1024$$

Specifying a Hysteresis Value

To convert a voltage value to a count when specifying a hysteresis value, use one of the following equations, where V_{hyst} is the desired voltage, and *span* is the appropriate value from Table B-1 on page B-2:

DAS-1700ST Series and DAS-1700AO Series Boards

$$\text{Count} = \frac{V_{\text{hyst}} \times 4096}{\text{span}}$$

DAS-1700HR Series Boards

$$\text{Count} = \frac{V_{\text{hyst}} \times 65536}{\text{span}}$$

Note: When converting voltage to counts to specify a hysteresis value, always use a gain of 1 to determine which span value to use from Table B-1, no matter what the gain of the channel.

For example, assume that you want to specify an analog trigger hysteresis value of 0.5 V for a channel on a DAS-1701ST board configured for a bipolar input range type. The count is determined as follows:

$$\frac{1.25\ \text{V} \times 4096}{10\ \text{V}} = 512$$

Index

A

about trigger 2-24
about-trigger acquisition 2-30
accessing a frame 2-6
accessing data
    C/C++ pointer indirection 3-5
    Pascal 3-14
    Visual Basic 3-19
allocating memory
    analog input operations 2-10
    analog output operations (AO) 2-44
    analog output operations (ST-DA, HR-DA) 2-35
    digital I/O operations 2-59
    dynamically in C/C++ 3-3
    dynamically in Pascal 3-11
    dynamically in Visual Basic for Windows 3-17
    locally in C/C++ 3-6
    locally in Pascal 3-14
    locally in Visual Basic for Windows 3-22
analog input operations 2-4
    programming flow diagrams 1-8
analog input ranges 2-12, 2-13
analog output channels: *see* channels
analog output operations (AO) 2-40
    programming flow diagrams 1-18
analog output operations (ST-DA, HR-DA) 2-32
    programming flow diagrams 1-15
analog output ranges (AO) 2-48
analog trigger
    analog input operations 2-25
    analog output operations (AO) 2-54
arrays, dimensioning and assigning
    C/C++ 3-6
    Pascal 3-14
    Visual Basic 3-22

B

board initialization 2-2
Borland C/C++
    programming information 3-10
    see also C languages
Borland Turbo Pascal for Windows: *see* Turbo Pascal for Windows
buffer address functions 4-3
buffer addresses
    analog input operations 2-12
    analog output operations (AO) 2-47
    analog output operations (ST-DA, HR-DA) 2-35
    digital I/O operations 2-59
buffering mode functions 4-3
buffering modes
    analog input operations 2-23
    analog output operations (AO) 2-53
    analog output operations (ST-DA, HR-DA) 2-39
    digital I/O operations 2-63
buffers
    analog input operations 2-10
    analog output operations (AO) 2-45
burst mode
    overview 2-18
    with SSH 2-19
burst mode conversion clock
    overview 2-22
    rates 2-22
    synchronizing operations (AO) 2-52

C

C languages
- accessing data 3-5
- creating a channel-gain queue 3-7
- dimensioning and assigning local arrays 3-6
- dynamically allocating memory buffers 3-3
- handling errors 3-8

channel and gain functions 4-4
channel-gain queue
- analog input operations 2-17
- analog output operations (AO) 2-49
- creating in C/C++ 3-7
- creating in Pascal 3-15
- creating in Visual Basic for Windows 3-23

channels
- analog input 2-14
- analog output (AO) 2-48
- both analog output using different ranges (AO) 2-49
- both analog output using the same range (AO) 2-48
- channel-gain queue (analog input operations) 2-17
- channel-gain queue (analog output operations, AO) 2-49
- digital input 2-60
- digital output 2-61
- group of consecutive channels 2-16
- single analog output (AO) 2-48

clock functions 4-4
clock: *see* burst mode conversion clock, pacer clock
commands: *see* functions
common-mode ground reference 2-14
continuous mode
- analog input operations 2-24
- analog output operations (AO) 2-53
- analog output operations (ST-DA, HR-DA) 2-39
- digital I/O operations 2-64

conventions 4-5
conversion mode functions 4-3
conversion modes 2-18
conversion rates
- analog input operations 2-21
- analog output operations (AO) 2-50

converting values
- 16-bit values (DAS-1700HR Series) 3-24
- counts to voltage B-1
- voltage to counts B-4

creating an executable file
- Borland C/C++ (for Windows) 3-10
- Microsoft C/C++ (for Windows) 3-9
- Turbo Pascal for Windows 3-16
- Visual Basic for Windows 3-26

D

D/A FIFO (AO) 2-42
DACs: *see* digital-to-analog converters
DAS-1700 Series Function Call Driver: *see* Function Call Driver
data transfer modes: *see* operation modes
data types 4-6
default values
- A/D frame elements 2-8
- D/A frame elements (AO) 2-43
- D/A frame elements (ST-DA, HR-DA) 2-34
- DI frame elements 2-58
- DO frame elements 2-59

device handles 2-2
digital I/O operations 2-56
    digital input programming flow diagrams 1-26
    digital output programming flow diagrams 1-29
digital trigger
    analog input operations 2-28
    analog output operations (AO) 2-54
digital-to-analog converters (AO) 2-48
digital-to-analog converters (ST-DA, HR-DA) 2-36
dimensioning local arrays
    C/C++ 3-6
    Pascal 3-14
    Visual Basic 3-22
DMA mode
    analog input operations 2-5
    analog output operations (AO) 2-41
driver handles 2-2
driver: *see* Function Call Driver
dual-DMA mode
    analog input operations 2-5
    analog output operations (AO) 2-41

E

error codes A-1
error handling 2-3
    C/C++ 3-8
    Pascal 3-16
    Visual Basic 3-25
EXP-1800 expansion accessory channels 2-14
external pacer clock
    analog input operations 2-21
    analog output operations (AO) 2-51

F

files required
    Borland C/C++ (for Windows) 3-10
    Microsoft C/C++ (for Windows) 3-9
    Turbo Pascal for Windows 3-16
    Visual Basic for Windows 3-25
flow diagrams 1-7
frame management functions 4-2
frames 2-6
    A/D 2-6
    D/A (AO) 2-42
    D/A (ST-DA, HR-DA) 2-34
    DI 2-58
    DO 2-58
    handles (analog input) 2-6
    handles (analog output, AO) 2-42
    handles (analog output, ST-DA, HR-DA) 2-34
    handles (digital I/O operations) 2-58
Function Call Driver
    initialization 2-2
functions
    buffer address 4-3
    buffer mode 4-3
    channel and gain 4-4
    clock 4-4
    conversion mode 4-3
    frame management 4-2
    gate 4-5
    initialization 4-2
    K_ADRead 2-4, 2-16, 4-7
    K_BufListAdd 2-12, 2-47, 4-9
    K_BufListReset 2-12, 2-47, 4-12
    K_ClearFrame 2-7, 4-14
    K_CloseDriver 2-2, 4-16
    K_ClrAboutTrig 4-18
    K_ClrADFreeRun 2-18, 4-20
    K_ClrContRun 2-23, 2-39, 2-53, 2-64, 4-22
    K_DASDevInit 2-3, 4-24

K_DAWriteGain 2-33, 2-36, 2-40, 2-48, 4-26
K_DIRead 2-56, 4-29
K_DMAAlloc 2-11, 2-46, 4-31
K_DMAFree 2-11, 2-46, 4-31, 4-34, 4-36, 4-38
K_DMAStart 2-5, 2-41, 4-36
K_DMAStatus 2-6, 2-41, 4-38
K_DMAStop 2-6, 2-41, 4-43
K_DOWrite 2-56, 4-46
K_FormatChnGAry 4-48
K_FreeDevHandle 2-2, 4-49
K_FreeFrame 2-7, 4-51
K_GetADCommonMode 4-52
K_GetADConfig 4-54
K_GetADFrame 2-6, 4-56
K_GetADMode 4-58
K_GetBurstTicks 4-60
K_GetClkRate 2-51, 4-62
K_GetDAFrame 2-34, 2-42, 4-64
K_GetDevHandle 2-2, 4-66
K_GetDIFrame 2-58, 4-68
K_GetDOFrame 2-58, 4-70
K_GetErrMsg 2-4, 4-72
K_GetShellVer 2-3, 4-73
K_GetVer 2-3, 4-75
K_IntAlloc 2-11, 2-35, 2-46, 2-60, 4-77
K_IntFree 2-11, 2-36, 2-46, 2-60
K_IntStart 2-5, 2-33, 2-41, 2-57, 4-81
K_IntStatus 2-5, 2-33, 2-41, 2-57, 4-83
K_IntStop 2-5, 2-33, 2-41, 2-57, 4-87
K_MoveArrayToBuf 2-36, 2-45, 2-60, 4-89
K_MoveBufToArray 2-10, 2-60, 4-91
K_OpenDriver 2-2, 4-93
K_RestoreChnGAry 4-95
K_SetAboutTrig 4-96
K_SetADCommonMode 2-14, 4-98
K_SetADConfig 2-14, 4-100
K_SetADFreeRun 2-18, 4-102
K_SetADMode 2-12, 4-104
K_SetADTrig 2-25, 4-106
K_SetBuf 4-108
K_SetBufI 4-111
K_SetBurstTicks 2-22, 4-113
K_SetChn 2-16, 2-36, 2-48, 4-115
K_SetChnGAry 2-18, 2-50, 4-117
K_SetClk 2-21, 2-22, 2-51, 4-120
K_SetClkRate 2-20, 2-37, 2-50, 2-62, 4-123
K_SetContRun 2-24, 2-39, 2-53, 2-64, 4-125
K_SetDITrig 2-28, 4-127
K_SetDMABuf 4-129
K_SetExtClkEdge 2-22, 2-51, 4-131
K_SetG 2-16, 2-17, 2-48, 4-133
K_SetGate 2-32, 2-55, 4-135
K_SetSSH 2-19
K_SetStartStopChn 2-16, 2-36, 2-49, 4-139
K_SetStartStopG 2-17, 2-48, 4-141
K_SetSync 2-53, 4-144
K_SetTrig 2-24, 2-28, 2-54, 4-146
K_SetTrigHyst 2-26, 4-148
memory management 4-3
miscellaneous 4-5
operation 4-2
summary 1-2
trigger 4-4

G

gain codes
    analog input operations 2-13
    analog output ranges (AO) 2-47
gains: *see also* analog input ranges 2-12
gate
    analog input operations 2-31
    analog output operations (AO) 2-55
gate functions 4-5
group of consecutive channels 2-16

H

handles
device 2-2
driver 2-2
frames (analog input) 2-6
frames (analog output, AO) 2-42
frames (analog output, ST-DA, HR-DA) 2-34
frames (digital I/O operations) 2-58
memory (analog input operations) 2-11
memory (analog output operations, AO) 2-46
hardware gate: *see* gate
help 1-31
hysteresis 2-26

I

initialization functions 4-2
initializing a board 2-2
initializing the driver 2-2
input channel configuration 2-14
input range type 2-12
internal A/D pacer clock
analog input operations 2-20
analog output operations (ST-DA, HR-DA) 2-37
digital I/O operations 2-62
synchronizing with the D/A clock (AO) 2-52
internal D/A pacer clock (AO) 2-50
internal trigger
analog input operations 2-24
analog output operations (AO) 2-54
interrupt mode
analog input operations 2-5
analog output operations (AO) 2-40
analog output operations (ST-DA, HR-DA) 2-33
digital I/O operations 2-57

K

K_ADRead 2-4, 2-16, 4-7
K_BufListAdd 2-12, 2-47, 4-9
K_BufListReset 2-12, 2-47, 4-12
K_ClearFrame 2-7, 4-14
K_CloseDriver 2-2, 4-16
K_ClrAboutTrig 4-18
K_ClrADFreeRun 2-18, 4-20
K_ClrContRun 2-23, 2-39, 2-53, 2-64, 4-22
K_DASDevInit 2-3, 4-24
K_DAWriteGain 2-33, 2-36, 2-40, 2-48, 4-26
K_DIRead 2-56, 4-29
K_DMAAlloc 2-11, 2-46, 4-31
K_DMAFree 2-11, 2-46, 4-31, 4-34, 4-36, 4-38
K_DMAStart 2-5, 2-41, 4-36
K_DMAStatus 2-6, 2-41, 4-38
K_DMAStop 2-6, 2-41, 4-43
K_DOWrite 2-56, 4-46
K_FormatChnGAry 4-48
K_FreeDevHandle 2-2, 4-49
K_FreeFrame 2-7, 4-51
K_GetADCommonMode 4-52
K_GetADConfig 4-54
K_GetADFrame 2-6, 4-56
K_GetADMode 4-58
K_GetBurstTicks 4-60
K_GetClkRate 2-51, 4-62
K_GetDAFrame 2-34, 2-42, 4-64
K_GetDevHandle 2-2, 4-66
K_GetDIFrame 2-58, 4-68
K_GetDOFrame 2-58, 4-70
K_GetErrMsg 2-4, 4-72
K_GetShellVer 2-3, 4-73
K_GetVer 2-3, 4-75
K_IntAlloc 2-11, 2-35, 2-46, 2-60, 4-77
K_IntFree 2-11, 2-36, 2-46, 2-60
K_IntStart 2-5, 2-33, 2-41, 2-57, 4-81
K_IntStatus 2-5, 2-33, 2-41, 2-57, 4-83

K_IntStop 2-5, 2-33, 2-41, 2-57, 4-87
K_MoveArrayToBuf 2-36, 2-45, 2-60, 4-89
K_MoveBufToArray 2-10, 2-60, 4-91
K_OpenDriver 2-2, 4-93
K_RestoreChnGAry 4-95
K_SetAboutTrig 4-96
K_SetADCommonMode 2-14, 4-98
K_SetADConfig 2-14, 4-100
K_SetADFreeRun 2-18, 4-102
K_SetADMode 2-12, 4-104
K_SetADTrig 2-25, 4-106
K_SetBuf 4-108
K_SetBufI 4-111
K_SetBurstTicks 2-22, 4-113
K_SetChn 2-16, 2-36, 2-48, 4-115
K_SetChnGAry 2-18, 2-50, 4-117
K_SetClk 2-21, 2-22, 2-51, 4-120
K_SetClkRate 2-20, 2-37, 2-50, 2-62, 4-123
K_SetContRun 2-24, 2-39, 2-53, 2-64, 4-125
K_SetDITrig 2-28, 4-127
K_SetDMABuf 4-129
K_SetExtClkEdge 2-22, 2-51, 4-131
K_SetG 2-16, 2-17, 2-48, 4-133
K_SetGate 2-32, 2-55, 4-135
K_SetSSH 2-19
K_SetStartStopChn 2-16, 2-36, 2-49, 4-139
K_SetStartStopG 2-17, 2-48, 4-141
K_SetSync 2-53, 4-144
K_SetTrig 2-24, 2-28, 2-54, 4-146
K_SetTrigHyst 2-26, 4-148

L

logical channel numbers 2-14

M

maintenance operations: *see* system operations
managing memory: *see* allocating memory
memory allocation: *see* allocating memory
memory buffers
  analog input operations 2-10
  analog output operations (AO) 2-45
memory handles
  analog input operations 2-11
  analog output operations (AO) 2-46
memory management functions 4-3
Microsoft C/C++ (for Windows)
  programming information 3-9
  see also C languages
Microsoft Visual Basic for Windows: *see* Visual Basic for Windows
miscellaneous functions 4-5
miscellaneous operations: *see* system operations
multiple memory buffers
  analog input operations 2-10
  analog output operations (AO) 2-45

O

operation functions 4-2
operation modes
  analog input operations 2-4
  analog output operations (AO) 2-40
  analog output operations (ST-DA, HR-DA) 2-33
  digital I/O operations 2-56
operations
  analog input 2-4
  analog output (AO) 2-40
  analog output (ST-DA, HR-DA) 2-32
  digital I/O 2-56
  system 2-1

P

paced mode 2-18
pacer clock
    analog input operations 2-19
    analog output operations (AO) 2-50
    analog output operations (ST-DA, HR-DA) 2-37
    digital I/O operations 2-62
Pascal
    accessing data 3-14
    creating a channel-gain queue 3-15
    dimensioning and assigning local arrays 3-14
    dynamically allocated memory buffers 3-11
    dynamically allocating memory buffers 3-11
    handling errors 3-16
physical channel numbers 2-14
post-trigger acquisition 2-29
preliminary procedures 1-7
pre-trigger acquisition 2-30
procedures 1-7
    analog input 1-8
    analog output (AO) 1-18
    analog output (ST-DA, HR-DA) 1-15
    digital input 1-26
    digital output 1-29
    preliminary 1-7
programming flow diagrams 1-7
programming information
    Borland C/C++ 3-10
    Microsoft C/C++ (for Windows) 3-9
    Turbo Pascal for Windows 3-16
    Visual Basic for Windows 3-17, 3-25
programming overview 3-2

R

recycle mode (AO) 2-42, 2-54
resetting a board 2-3
return values 2-3
revision levels 2-3
routines: *see* functions

S

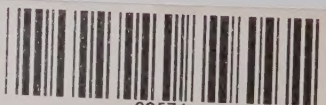
scan 2-16
single mode
    analog input operations 2-4
    analog output operations (AO) 2-40
    analog output operations (ST-DA, HR-DA) 2-33
    digital I/O operations 2-56
single-cycle mode
    analog input operations 2-23
    analog output operations (AO) 2-53
    analog output operations (ST-DA, HR-DA) 2-39
    digital I/O operations 2-64
software trigger: *see* internal trigger
start trigger 2-24
starting analog input operations 2-4
starting analog output operations (AO) 2-40
starting analog output operations (ST-DA, HR-DA) 2-33
starting digital I/O operations 2-56
status codes 2-3, A-1
summary of functions 1-2
synchronizing D/A conversions (AO) 2-52
system operations 2-1

T

tasks 1-7
  analog input 1-8
  analog output (AO) 1-18
  analog output (ST-DA, HR-DA) 1-15
  digital input 1-26
  digital output 1-29
  preliminary 1-7
technical support 1-31
time base
  burst mode conversion clock 2-22
  internal A/D pacer clock 2-20
trigger functions 4-4
triggers
  analog input operations 2-24
  analog output operations (AO) 2-54
troubleshooting 1-31
Turbo Pascal for Windows
  programming information 3-16
  see also Pascal

U

update rate
  DAS-1700AO Series 2-50
  DAS-1700ST-DA Series and DAS-1702HR-DA 2-37
  see also conversion rate
using 16-bit values (DAS-1700HR Series)
  Visual Basic for Windows 3-24

V

Visual Basic for Windows
  accessing data 3-19
  creating a channel-gain queue 3-23
  dimensioning and assigning local arrays 3-22
  dimensioning arrays 3-22
  dynamically allocating memory buffers 3-17
  handling errors 3-25
  programming information 3-17, 3-25